

INITIATION AU LANGAGE ASSEMBLEUR

B. GEOFFRION - H. LILEN



S. E. C. F.



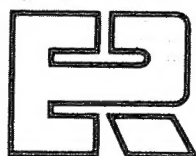
EDITIONS RADIO

**INITIATION
AU LANGAGE
ASSEMBLEUR**

B. GEOFFRION - H. LILEN

INITIATION AU LANGAGE ASSEMBLEUR

S. E. C. F.



Editions Radio

9, RUE JACOB - 75006 PARIS
TÉL. 329.63.70

La loi du 11 mars 1957 n'autorisant, aux termes des alinéas 2 et 3 de l'article 41, d'une part, que les « copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective » et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, « toute représentation ou reproduction intégrale ou partielle, faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause, est illicite » (alinéa 1^{er} de l'article 40). Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait donc une contrefaçon sanctionnée par les articles 425 et suivants du Code pénal.

<p>© SECF Éditions Radio, Paris 1983</p> <p><i>Tous droits de traduction, de reproduction et d'adaptation réservés pour tous pays</i></p>	<p>Imprimerie Berger-Levrault, Nancy</p> <p>Dépôt légal : décembre 1983 Éditeur n° 951 - Imprimeur : 778517 I.S.B.N. 2 7091 0935 2</p>
---	--

PREAMBULE

Les «50 Programmes» qui suivent s'adressent à tous les possesseurs d'un ordinateur construit autour de l'un des microprocesseurs suivants : 8080, 8085, Z 80, NSC 800. Il sont écrits en langage assembleur 8085.

Ils sont généralement courts, moins de 200 octets et font appel à la majorité des instructions du 8085, certains ayant été plus particulièrement conçus pour montrer les utilisations de telle ou telle instructions et les pièges éventuels.

Les programmes sont fournis avec l'organigramme du problème à résoudre et le listage (code machine et assembleur) avec commentaires.

Il est vivement conseillé aux débutants de suivre la progression du livre même si le sujet traité ne les inspire pas. Ils acquerront ainsi les techniques leur permettant de maîtriser les règles fondamentales de la programmation.

Les auteurs ne prétendent pas avoir écrit, pour chaque cas envisagé, le meilleur programme, leur but étant de fournir au lecteur les bases indispensables pour résoudre ses propres problèmes.

Les premiers programmes sont très détaillés pour permettre aux débutants de comprendre les instructions et les raisons du mode opératoire, les suivants vont directement au but, chacun d'entre-eux ayant une raison précise : introduction de nouvelles instructions montrant leur emploi, initiation à de nouveaux principes. Quelques jeux facilitent l'assimilation de l'ensemble.

Les auteurs souhaitent ainsi que les lecteurs aient autant d'intérêt que de satisfaction à étudier ce livre et à travailler avec les microprocesseurs et les micro-ordinateurs.

INTRODUCTION

Les «50 Programmes» ont été écrits pour le système minimal : le kit SDK 85. Certains d'entre-eux font appel aux programmes du moniteur - entrée clavier, affichage - ils ne pourront donc être utilisés dans leur intégralité que si vous connaissez parfaitement votre ordinateur qui doit vous être fourni avec **toute** la documentation nécessaire ; sinon harcelez le vendeur.

Nous vous conseillons, toutefois, de les étudier car ils vous serviront d'exemple pour certaines instructions ou pour construire votre propre micro-ordinateur.

Les utilisateurs de micro-ordinateurs n'auront dans la majorité des cas que quelques modifications à apporter dépendant du système utilisé. Par exemple l'instruction «FIN» qui rend la main au moniteur peut être remplacée par un ordre du type GO suivi de l'adresse de la dernière instruction ce qui correspond à un point d'arrêt permettant l'examen des cases mémoires et des registres.

Nous commencerons cet ouvrage par un aperçu des quatre microprocesseurs 8 bits 8080, 8085, Z 80 et NSC 800 en insistant sur leurs similitudes et leurs différences. Les 50 programmes suivront. Enfin en appendices nous détaillerons le kit SDK 85 en explicitant l'usage des touches du clavier et le travail à la console ou sur téléimprimeur. Nous invitons les lecteurs qui désireraient plus d'informations sur les instructions des microprocesseurs 8080/8085 et Z 80 à se reporter aux ouvrages de L.A. Leventhal édités en français aux Editions Radio.

POUR MIEUX SUIVRE LES LISTAGES

Les programmes proposés sont présentés sous forme de listage («Listing», en anglais) issu d'un système de développement. Les auteurs en ayant utilisé deux, L'Intellec de Intel et le Pace de National Semiconductors, les listages pourront apparaître différents *dans leur forme et leur présentation* tout au long de cet ouvrage.

La machine se présente, en première ligne : «ISIS-II 8080/8085 Macro-assembleur, V4.0» avec, pour titre «Exemple page 1».

Les têtes des colonnes sont : *LOC*, pour «location», adresse ; *OBJ* pour code objet ; *LINE* pour le numéro de la ligne ; *SOURCE STATEMENT* pour code source. Le nom du programme (*NAME*) est donné avec ici «EXEMPLE 1». Sur l'autre machine, on trouvera *TITLE* (titre).

On trouvera également dans ces listages des *pseudo-instructions*, ou *directives*, servant uniquement à la machine pour exécuter l'assemblage : *ORG* pour lui indiquer l'adresse d'*origine* du programme, souvent 2000 dans les exemples ; *EQU* pour les *équivalences* : à une donnée est substitué une étiquette ; *END* pour lui préciser que le programme est terminé ; etc. Toutes ces pseudo-instructions sont définies dans les manuels d'emplois des systèmes de développement.

LE MATERIEL

Dans ce chapitre nous rappellerons les structures des quatre microprocesseurs et leur jeu d'instructions. Pour plus de détails nous vous conseillons de vous reporter aux notices des constructeurs.

Les quatre microprocesseurs ont en commun les instructions du 8080 dont le brochage est donné par la figure 1 et la structure par la figure 2.

8080

8080

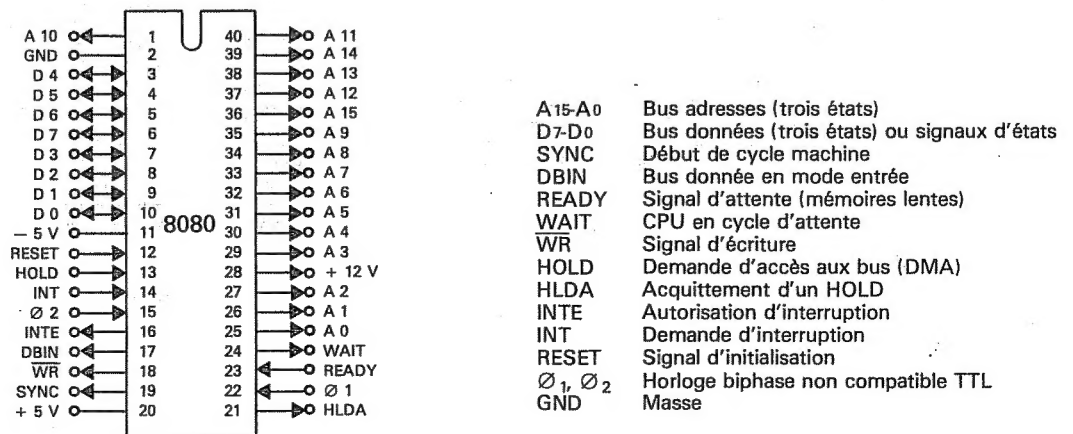


figure 1 : brochage du 8080

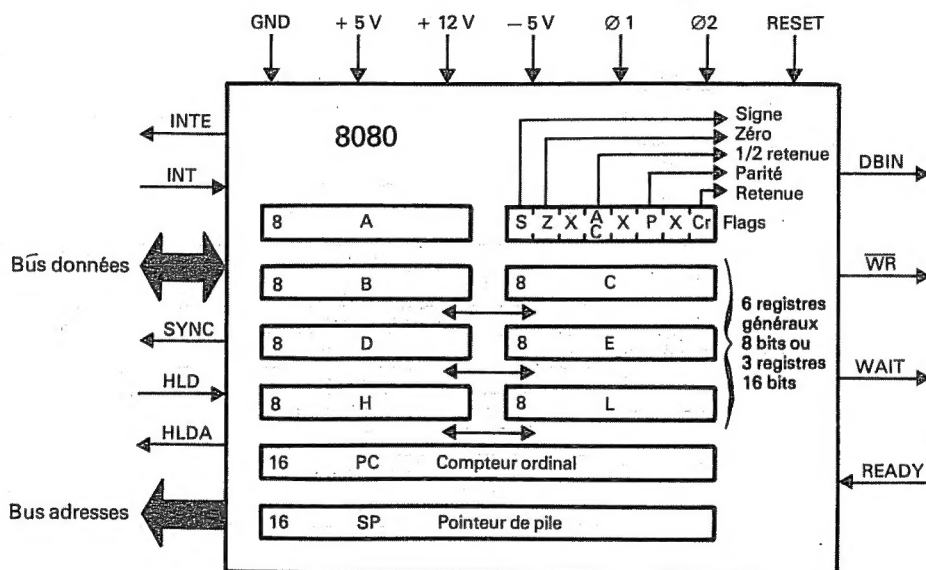


figure 2 : structure du 8080

Le tableau I nous fournit le jeu d'instructions en mnémonique et le code hexadécimal. En associant le quartet (4 bits) de poids fort (H) au quartet de poids faible (B). Les codes 10, 18, 20, 28, 30, 38, CB, D9, DD, E9 et FD ne correspondent à aucune instruction.

8085

Le brochage est donné par la figure 3 et la structure par la figure 4. Ce microprocesseur est monotension :

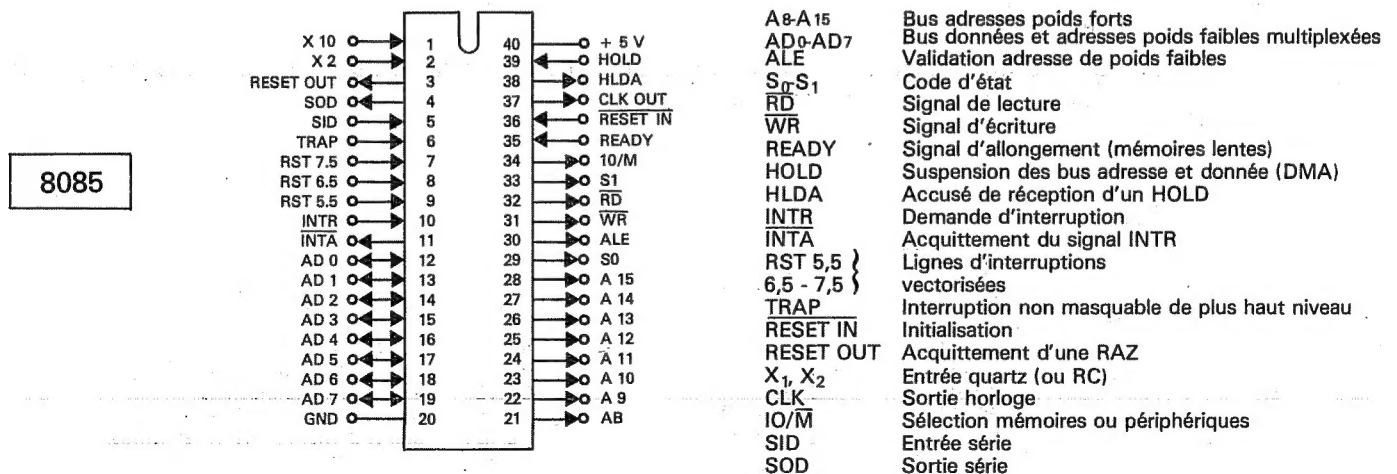


Figure 3 : brochage du 8085

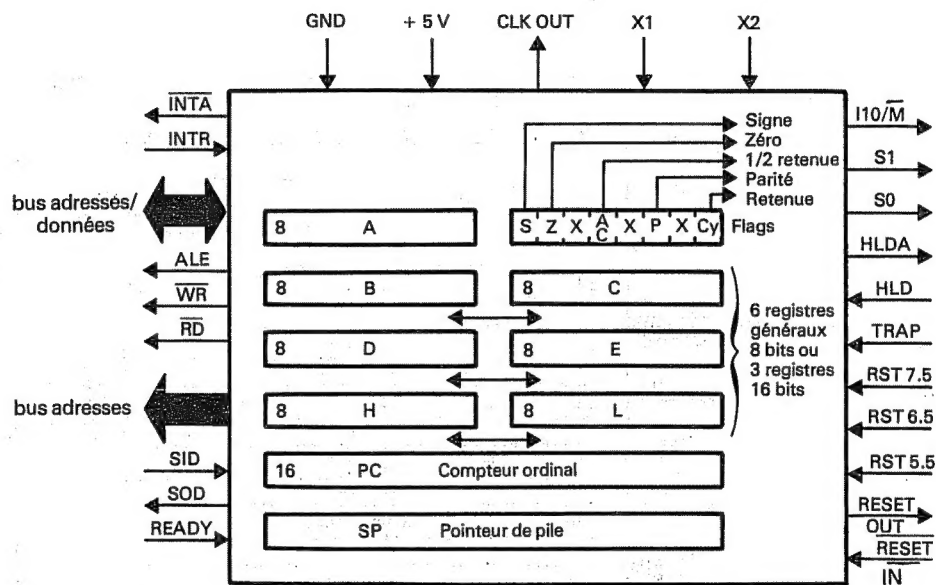


Figure 4 : structure du 8085

Tableau I - Instruction du 8080

H/B	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP	LXI	B STAX	B INX	B INR	B DCR	B MVI	B RLC		DAD	B LDAX	B DCX	B INR	C DCR	C MVI	C RRC
1	LXI	LXI	D STAX	D INX	D INR	D DCR	D MVI	D RAC		DAD	D LDAX	D DCX	D INR	E DCR	E MVI	E RAR
2	LXI	LXI	H SHLD	H INX	H INR	H DCR	H MVI	H DAA		DAD	H LHLD	H DCX	H INR	L DCR	L MVI	L CMA
3	LXI	LXI	SP STAX	SP INX	SP INR	M DCR	M MVI	M STC		DAD	SP LDA	DCX	SP INR	A DCR	A MVI	A CMC
4	MOV	BB MOV	BC MOV	BD MOV	BE MOV	BH MOV	BL MOV	BM MOV	CB MOV	CB MOV	CC MOV	CD MOV	CE MOV	CH MOV	CL MOV	CM MOV
5	MOV	DB MOV	DC MOV	DD MOV	DE MOV	DH MOV	DL MOV	DM MOV	DA MOV	EB MOV	EC MOV	ED MOV	EE MOV	EH MOV	EL MOV	EM MOV
6	MOV	HB MOV	HC MOV	HD MOV	HE MOV	HH MOV	HL MOV	HM MOV	HA MOV	LB MOV	LC MOV	LD MOV	LE MOV	LH MOV	LL MOV	LM MOV
7	MOV	MB MOV	MC MOV	MD MOV	ME MOV	MH MOV	ML HLT	M MOV	MA MOV	AB MOV	AC MOV	AD MOV	AE MOV	AH MOV	AL MOV	AM MOV
8	ADD	B ADD	C ADD	D ADD	E ADD	H ADD	L ADD	M ADD	A ADD	B ADD	C ADD	D ADD	E ADD	H ADD	L ADD	M ADD
9	SUB	B SUB	C SUB	D SUB	E SUB	H SUB	L SUB	M SUB	A SUB	B SUB	C SUB	D SUB	E SUB	H SUB	L SUB	M SUB
A	ANA	B ANA	C ANA	D ANA	E ANA	H ANA	L ANA	M ANA	A ANA	B ANA	C ANA	D ANA	E ANA	H ANA	L ANA	M ANA
B	ORA	B ORA	C ORA	D ORA	E ORA	H ORA	L ORA	M ORA	A ORA	B ORA	C ORA	D ORA	E ORA	H ORA	L ORA	M ORA
C	RNZ	POP	B JNZ	adr JMP	adr CNZ	adr PUSH	B ADI	RST	0 RZ	RET	JZ	adr IN	CZ	adr CALL	adr ACI	RST
D	RNC	POP	D JNC	adr OUT	adr CNC	adr PUSH	D SUI	RST	2 RZ		JC	adr IN	adr XCHG	adr CPE	SBI	RST
E	RPO	POP	H JPO	adr XTHL	CPO	adr PUSH	H ANI	RST	4 RPE		PCHL	adr IN	adr XCHG	adr CPE	adr SBI	RST
F	RP	POP	PSW JP	adr DI	CP	adr PUSH	PSW ORI	RST	6 RM	SPHL	JM	adr EI	CM	adr	CPI	RST

Le code machine est obtenu en associant H et B

Exemple : 01 est le code de LXI B

Les instructions de chargement immédiat du type LXI B au MOV A exigent de faire suivre le code de l'instruction

de deux octets pour la première et de un octet pour la seconde. Les instructions concernant les opérations arithmétiques ou logiques immédiates du type ANI ou ORI sont à deux octets : le code suivi de la donnée.

Tableau II : Instructions du 8085



H/B	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP	LXI	B STAX	B INX	B INR	B DCR	B MVI	B RLC		DAD	B LDAX	B DCX	B INR	C DCR	C MVI	C RRC
1	LXI	LXI	D STAX	D INX	D INR	D DCR	D MVI	D RAL		DAD	D LDAX	D DCX	D INR	E DCR	E MVI	E RAR
2	LXI	LXI	H SHLD	H INX	H INR	H DCR	H MVI	H DAA		DAD	H LHLD	H DCX	H INR	L DCR	L MVI	L CMA
3	LXI	LXI	SP STA	SP INX	SP INR	M DCR	M MVI	M STC		DAD	SP LDA	DCX	SP INR	A DCR	A MVI	A CMC
4	MOV	BB MOV	BC MOV	BD MOV	BE MOV	BH MOV	BL MOV	BM MOV	BA MOV	CB MOV	CC MOV	CD MOV	CE MOV	CH MOV	CL MOV	CM MOV
5	MOV	DB MOV	DC MOV	DD MOV	DE MOV	DH MOV	DL MOV	DM MOV	DA MOV	EB MOV	EC MOV	ED MOV	EE MOV	EH MOV	EL MOV	EM MOV
6	MOV	HB MOV	HC MOV	HD MOV	HE MOV	HH MOV	HL MOV	HM MOV	HA MOV	LB MOV	LC MOV	LD MOV	LE MOV	LH MOV	LL MOV	LM MOV
7	MOV	MB MOV	MC MOV	MD MOV	ME MOV	MH MOV	ML HLT	M MOV	MA MOV	AB MOV	AC MOV	AD MOV	AE MOV	AH MOV	AL MOV	AM MOV
8	ADD	B ADD	C ADD	D ADD	E ADD	H ADD	L ADD	M ADD	A ADD	B ADD	C ADD	D ADD	E ADD	H ADD	L ADD	M ADD
9	SUB	B SUB	C SUB	D SUB	E SUB	H SUB	L SUB	M SUB	A SUB	B SUB	C SUB	D SUB	E SUB	H SUB	L SUB	M SUB
A	ANA	B ANA	C ANA	D ANA	E ANA	H ANA	L ANA	M ANA	A ANA	B ANA	C ANA	D ANA	E ANA	H ANA	L ANA	M ANA
B	ORA	B ORA	C ORA	D ORA	E ORA	H ORA	L ORA	M ORA	A ORA	B ORA	C ORA	D ORA	E ORA	H ORA	L ORA	M ORA
C	RNZ	POP	B JNZ	adr JMP	adr CNZ	adr PUSH	B ADI	RST	0 RZ	RET	JZ	adr IN	CZ	adr CALL	adr ACI	RST
D	RNC	POP	D JNC	adr OUT	adr CNC	adr PUSH	D SUI	RST	2 RZ		JC	adr IN	adr XCHG	adr CPE	SBI	RST
E	RPO	POP	H JPO	adr XTHL	CPO	adr PUSH	H ANI	RST	4 RPE		PCHL	adr IN	adr XCHG	adr CPE	adr SBI	RST
F	RP	POP	PSW JP	adr DI	CP	adr PUSH	PSW ORI	RST	6 RM	SPHL	JM	adr EI	CM	adr	CPI	RST

Ici les codes 20 et 30 sont utilisés et correspondent aux instructions RIM et SIM concernant l'entrée et la sortie série (S.I.D et S.O.D) d'une part et les interruptions vectorisées (RST 5.5 - RST 6.5 et RST 7.5) d'autre part. Les autres codes ne sont toujours pas définis, pourtant W. Dehnhart et V.M. Sorensen publient dans «Electronics» de janvier 1979 les opérations que de tels

codes entraînent bien qu'ils ne soient pas garantis par INTEL. Nous les donnons, ci-après, (Tableau III) à vous de tester votre 8085. Deux indicateurs apparaissent X 5 (entre Z et AC) indique le passage de FFFF à 0000 et vice-versa lors de l'exécution d'instruction comme INX ou DCX et Y (entre P et CY) indique le dépassement lors de la complémententation à 2.

Tableau II
Instructions du 8085

Tableau III

Code	Mnémonique	Opération
08	DSUB	$(HL) = (HL) - (BC)$
10	ARHL	 décalage de 1 bit à droite avec $H_7 = H_7$ et $Cy = L_0$
18	RDEL	 décalage de 1 bit à gauche avec $E_0 = Cy$ puis $Cy = D_7$
28 (2)	LDHI, donnée 16 bits	$(DE) = (HL) + \text{donnée 16}$
38 (2)	LDSI, donnée 26 bits	$(DE) = (SP) + \text{donnée 16}$
CB	RST V	si $V = 1$ (bit 1) $PC = 40$ Hexadécimal
D9	SHLX	$((DE)) = (L)$ $((DE) + 1) = (H)$
DD (2)	JNX5, adresse	si $X5 = 0$ (bit 5) $PC = \text{adresse}$
ED	LHLX	$(L) = ((DE))$ $(H) = ((DE) + 1)$
FD (2)	JX5, adresse	si $X5 = 1$ $PC = \text{adresse}$

Instructions «cachées» du 8085. Le chiffre 2 entre parenthèses indique qu'il faut faire suivre, en langage machine, le code instruction de deux octets.

Z 80

Le brochage et la structure sont donnés par les figures 5 et 6

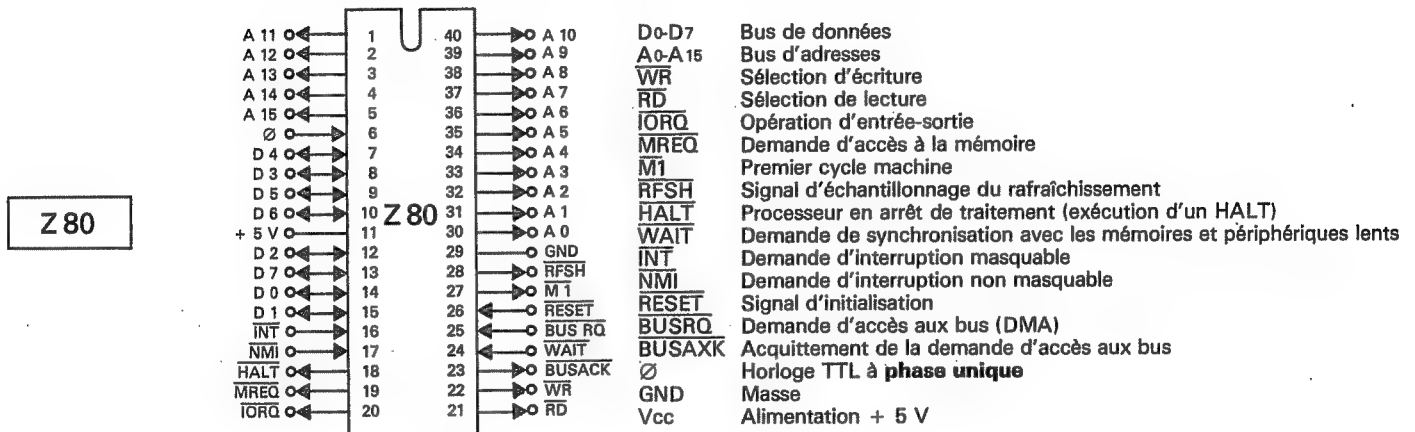


Figure 5 : brochage du Z 80

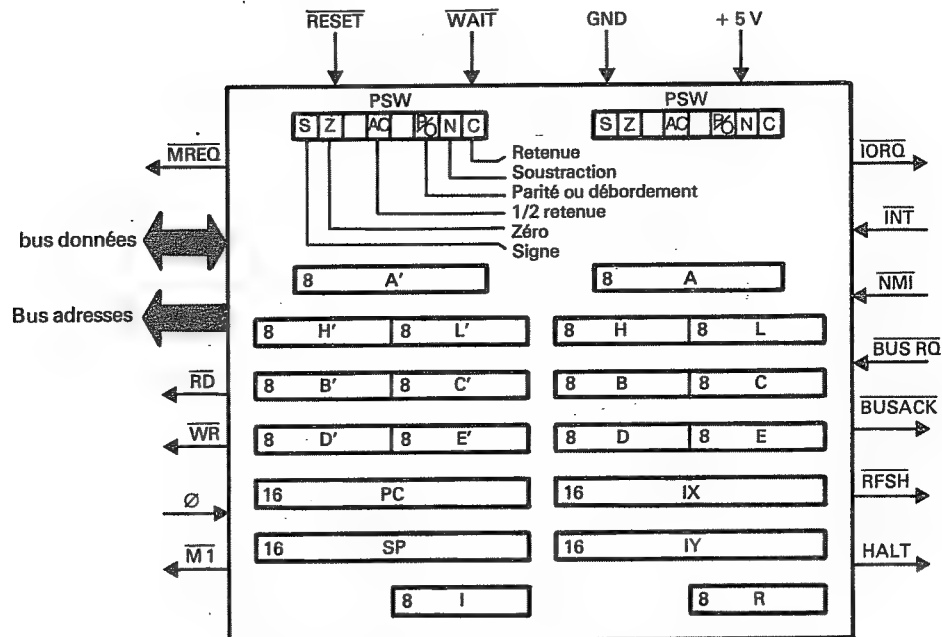


Figure 6 : structure du Z 80

Ici le nombre de registres a été doublé, les instructions permettent de passer d'un jeu à l'autre. Le registre R sert au rafraîchissement des mémoires dynamiques. Le registre I concerne les pages d'adresses des interruptions et des deux registres. IX et IY sont utilisés lors d'adressages indexés.

Le tableau IV fournit les instructions du Z80. On constate que les codes inutilisés du 8080 sont ici largement employés. Certaines instructions pouvant nécessiter 4 octets.

Les mnémoniques de Z80 sont différents des mnémoniques de 8080 ou 8085 mais les codes machines pour une même opération sont identiques. Par exemple le résultat de l'exécution LXI B est identique à celui de l'exécution LD BC et le code de l'instruction est 01 pour les deux microprocesseurs. Un programme écrit en langage machine pour 8080 ou 8085 - sans SIM et RIM - tourne avec un Z80.

Tableau IV : Instructions du Z 80

Code	Mnémonique	Code	Mnémonique	Code	Mnémonique	Code	Mnémonique	Code	Mnémonique
00	NOP	56	LD	D.(HL)	RST	10H	RRD	ED 67	
01 yyyy	LD BD, data 16	57 1sss	LD	E, reg	RET	C	RLD	ED 6F	
02	LD (BC), A	5E	LD	E.(HL)	EXX		LDI	ED A0	
03	INC BC	6 0sss	LD	H, reg	JP	C, addr	CPI	ED A1	
04	INC B	66	LD	H.(HL)	IN	A, (port)	INI	ED A2	
05	DEC B	6 1sss	LD	L, reg	CALL	C, addr	OUTI	ED A3	
06 yy	LD B, data	6E	LD	L.(HL)	ADD	IX, pp	LDD	ED A8	
07	RLCA	7 0sss	LD	(HL), reg	LD	IX, data 16	CPD	ED A9	
08	EX AF, AF'	76	HALT		LD	(addr), IX	IND	ED AA	
09	ADD HL, BC	7 0sss	LD	A, reg	INC	IX	OUTD	ED AB	
0A	LD A.(BC)	7E	LD	A.(HL)	LD	IX (addr)	LDIR	ED B0	
0B	DEC BC	8 0rrr	LD	A, reg	DEC	IX	CPRI	ED B1	
0C	INC C	86	ADD	A.(HL)	INC	(IX+disp)	INIR	ED B2	
0D	DEC C	8E 1rrr	ADD	A.reg.	DEC	(IX+disp) data	OTIR	ED B3	
0E yy	LD C, data	8E	ADC	A.(HL)	LD	reg.(IX+disp) reg	LDDR	ED B8	
0F	RRCA	9 0rrr	SUB	reg	LD	(IX+disp) reg	CPDR	ED B9	
10 disp-2	LD D, disp	96	SUB	(HL)	LD	A.(IX+disp)	INDR	ED BA	
11 yyyy	LD DE, data 16	9 1rrr	SBC	A.reg.	LD	A.(IX+disp)	XOR	data	
12	LD (DE), A	9E	SBC	A.(HL)	LD	(IX+disp)	RST P	28H	
13	INC DE	A 0rrr	AND	reg	ADC	(IX+disp)	RET P		
14	INC D	A6	AND	(HL)	SUB	(IX+disp)	POP	AF	
15	DEC D	A 1rrr	XOR	reg.	DD 9E disp	(IX+disp)	IP	P.addr	
16 yy	LD D, data	AE	XOR	(HL)	DD A6 disp	(IX+disp)	DI		
17	RLA	B 0rrr	OR	reg.	DD AE disp	(IX+disp)	PUSH	P.addr	
18 disp-2	LD RLA, disp	B6	OR	(HL)	DD B6 disp	(IX+disp)	OR	AF	
19	ADD HL, DE	B 1rrr	CP	reg.	DD BE disp	(IX+disp)	CALL	AF	
1A	LD A.(DE)	BE	CP	(HL)	DD CB disp 06	(IX+disp)	PUSH	data	
1B	DEC DE	C0	CF	(HL)	DD CB disp 0E	(IX+disp)	RST 30H		
1C	INC E	C1	RET	NZ	DD CB disp 16	(IX+disp)	RET M		
1D	DEC E	C2 ppqq	POP	BC	DD CB disp 1E	(IX+disp)	LD SP, HL		
1E yy	LD E, data	C3 ppqq	JP	NZ, addr	DD CB disp 26	(IX+disp)	JP	M.addr	
1F	RRR	C4 ppqq	CALL	addr	DD CB disp 2E	(IX+disp)	EI		
20 disp-2	LD RRR, NZ, disp	C5	PUSH	NZ, addr	DD CB disp 3E	(IX+disp)	CALL	M.addr	
21 yyyy	LD HL, data 16	C6 yy	ADD	A, data	DDCBdisp010bb110	(IX+disp)	FC ppqq		
22 ppqq	LD (addr), HL	C7	RST	00H	DDCBdisp10bb110	(IX+disp)	FD 00rr9		
					DDCBdisp11bb110	(IX+disp)	FD 21 yyyy		

Tableau IV : Instructions du Z 80 (suite)

Code	Mnémoniques	Code	Mnémoniques	Code	Mnémoniques	Code	Mnémoniques	Code	Mnémoniques
23	INC HL	C8	RET Z	DD E1	POP IX	FD 22 ppqq	LD (addr),ly		
24	IND H	C9	RET	DDE3	EX (SP),IX	FD 23	INC IX		
25	DEC H	CA ppqq	JP Z.addr	DD E5	PUSH IX	FD 2A ppqq	LD IX(addr)		
26 yy	LD H.data	CB 0 0rrr	RLC reg	DD E9	JP (IX)	FD 2B	DEC IX		
27	DAA	CB 06	RLC (HL)	DD F9	LD SP,IX	FD 34 disp	INC (IX+disp)		
28 disp-2	IR Z,disp	CB 0 1rrr	RRC reg.	DE yy	SBC A.data	FD 35 disp	DEC (IX+disp)		
29	ADD HL,HL	CB 0E	RRC (HL)	DF	RST 18H	FD 36 disp yy	LD (IX+disp),data		
2A ppqq	LD HL,(addr)	CB 1 0rrr	RL reg.	EO	RET PO	FD 01ddd110disp	LD reg.(IX+disp)		
2B	DEC HL	CB 16	RL (HL)	E1	POP HL	FD 7 0sss disp	LD (IX+disp),reg.		
2C	INC L	CB 1 1rrr	RR reg.	E2 ppqq	JP PO,addr	FD 86 disp	ADD A (IX+disp)		
2D	DEC L	CB 1E	RR (HL)	E3	EX (SP),HL	FD 8E disp	ADC A.(IX+disp)		
2E	LD L.data	CB 2 0rrr	SLA reg.	E4 ppqq	CALL PO,addr	FD 96 disp	SUB A.(IX+disp)		
2F	CPL	CB 26	SLA (HL)	E5	PUSH HL	FD 96 disp	SBC A.(IX+disp)		
30 disp-2	IR NC,disp	CB 2 1rrr	SRA reg.	E6 yy	AND data	FD A6 disp	AND (IX+disp)		
31 yyyy	LD SP,data 16	CB 2E	SRA (HL)	E7	RST 20H	FD AE disp	XOR (IX+disp)		
32 ppqq	LD SP,Sladdr/A	CB 3 1rrr	SRL reg.	E8	RET PE	FD B6 disp	OR (IX+disp)		
33	INC SP	CB 3E	SRL (HL)	E9	JP (HL)	FD BE disp	CP (IX+disp)		
34	INC (HL)	CB 01brrr	BIT b.reg.	EA ppqq	JP PE,addr	FC CB disp 06	RLC (IX+disp)		
35	DEC (HL)	CB 01bbb110	BIT b.(HL)	EB	EX DE,HL	FD CB disp 0E	RRC (IX+disp)		
36 yy	LD (HL),data	CB 01bbb110	RES b.reg.	EC ppqq	CALL PE,addr	FC CB disp 16	RL (IX+disp)		
37	SCF	CB 10bbb110	RES b.(HL)	ED 01ddd000	IN reg.(C)	FD CB disp 1E	RR (IX+disp)		
38	IR C,disp	CB 11brrr	SET b.reg.	ED 01sss001	OUT (C),reg.	FD CB disp 26	SLA (IX+disp)		
39	ADD HL,SP	CB 11bbb110	SET b.(HL)	ED 01rrx2	SBC (HL),rp	FD CB disp 2E	SRA (IX+disp)		
3A ppqq	LD A,(addr)	CC ppqq	CALL Z,addr	ED 01xx3 ppqq	LD (addr),rp	FD CB disp 3E	SRL (IX+disp)		
3B	DEC SP	CC ppqq	CALL addr	ED 44	NEG	FD CBdisp01bbb110	BIT b.(IX+disp)		
3C	INC A	CE yy	ADC A.data	ED 45	RETN	FD CBdisp10bbb110	RES b.(IX+disp)		
3D	DEC A	CF	RST H	ED 010nm110	IM m	FD CBdisp11bbb110	SET b.(IX+disp)		
3E yy	LD A.data	D0	RET NC	ED 47	LD 1,A	FD E1	POP IX		
3F	CCF	D1	POP DE	ED 01xx A	ADC HL,rp	FD E3	EX (SP),IX		
4 0sss	LD B.reg.	D2 ppqq	JP NC,addr	ED 01xx B ppqq	LD rp,(addr)	FD E5	PUSH IX		
46	LD B.(HL)	D3 yy	OUT (port),A	ED 4D	RETI	FD E9	JP (IX)		
4 Iss	LD C.reg.	D4 ppqq	CALL NC,addr	ED 4F	LD R,A	FD F9	LD SP,IX		
4E	LD C.(HL)	D5	PUSH DE	ED 57	LD A,1	FE yy	CP data		
LD	D.reg.	D6 yy	SUB data	ED 5F	LD A,R	FF	RST 38H		

NSC 800

Le brochage est donnée par la figure 7

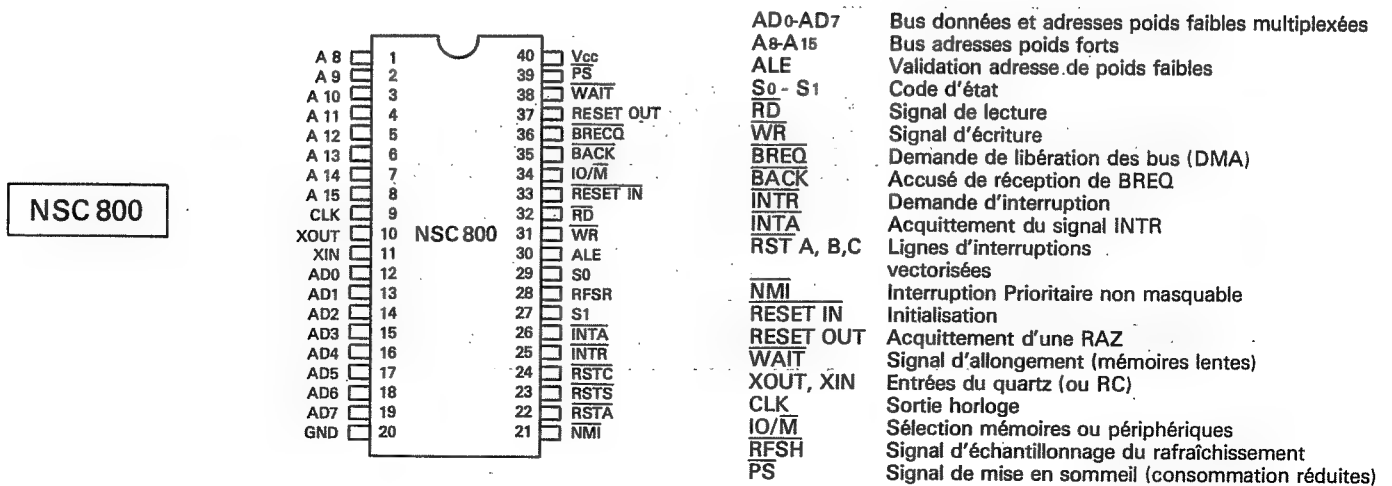


Figure 7 : brochage du NSC 800

La structure interne et les instructions sont celles du Z 80.

C'est un circuit du type CMOS. La tension d'alimentation peut donc être comprise entre 3 et 12 volts et la consommation est d'environ 50 mW sous 5 volts.

1

MULTIPLICATION DECIMALE DE DEUX NOMBRES DE UN CHIFFRE (DIFFERENTS DE ZERO)

But : Développer l'algorithme de base de la multiplication, apprendre à utiliser le pas à pas pour corriger un programme erroné.

Principales et nouvelles instructions utilisées : La touche SINGLE STEP, MOV et MVI, LXI.

On se propose d'écrire un programme réalisant la multiplication de deux chiffres décimaux (0 à 9) ; pour ce faire, nous utiliserons l'algorithme suivant :

$$P = a \times b = \sum_{1}^b a$$

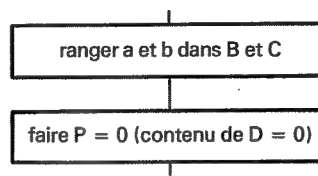
c'est-à-dire que l'on effectuera l'opération :

$$P = a + a + \dots + a$$

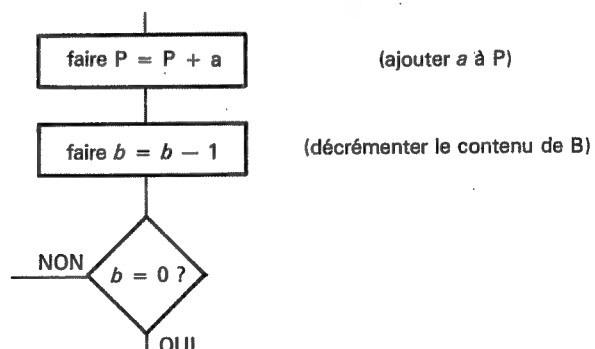
avec b fois a .

Pour écrire l'organigramme, il faut choisir un mode de travail. L'existence de registres internes et d'instructions inter-registres nous invite à l'utilisation de ces derniers.

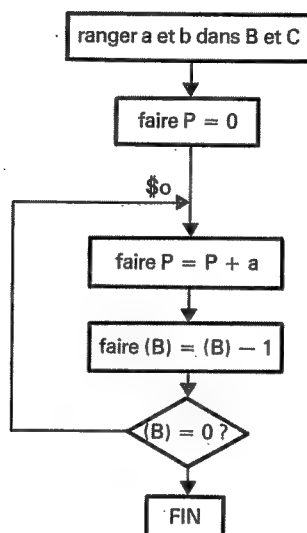
Il ne faut pas oublier que le contenu initial des mémoires (registre y compris) est indéterminé : il faudra *initialiser*. On choisit ici les registres B et C pour a et b et D pour le produit, c'est-à-dire que l'on mettra a et b dans B et C, par programme et qu'on ira lire le résultat dans D, ce qui donne :



Nous voulons faire $a + a + \dots + a$, b fois. En tenant compte des instructions du 8085 nous voyons que l'instruction DCRr affecte les indicateurs («*flags*») en particulier le flag Z qui est mis à 1 («*set*») si le résultat de l'opération est nul. Nous allons donc, à chaque fois que nous ferons une addition, soustraire 1 (décrémenter) au contenu au registre contenant b (B) et tester grâce à l'indicateur Z, l'arrivée à 0 du contenu de B, donc la fin de l'opération :



Après le test il y a deux possibilité $b = 0$ et $b \neq 0$. Si $b = 0$, nous avons terminé ; si b et différent de 0, il faut revenir à $P = P + a$, donc remonter dans l'organigramme pour faire une nouvelle boucle, le point de retour sera noté $\$0$. Notre organigramme est donc le suivant :



Comment écrire le programme avec les instructions du 8085 ?

1. — *Ranger a et b dans B et C* : on prend l'instruction LXI B qui permet de ranger deux nombres dans B et C (en langage machine inversée B et C)

LXI B, b a

soit «01 ab» : a sera mis dans C et b dans B.

2. — *Faire $P = 0$* , donc MVI D, 00 soit 16 00.

3. — *Faire $P = P + a$* .

Les opérations se font dans l'accumulateur A, il faut donc amener P (ou a) dans A, additionner dans A les contenus de C (a) et D (P) et remettre le résultat dans D, Ce qui donne :

```

MOV  A,D  P est mis dans A mais est conservé dans D
ADD  C    A contient P + a
MOV  D,A  le nouveau P est rangé dans D
  
```

ou

```

MOV  A,C  79
ADD  D    82
MOV  D,A  57
  
```

N.B. — On utilise ADD et non ADC pour éviter d'être perturbé par un CY («carry») éventuel.

4. — *Faire $b = b - 1$* donc DCR B, soit 05.

5. — *$b = 0$?* il suffit d'écrire JNZ \$0 (C2 \$0) ; si l'indicateur Z n'est pas à 1, c'est-à-dire si le contenu de B n'est pas nul, le microprocesseur reviendra en \$0 ; pour ce faire, l'adresse du «point» \$0 sera mise dans le compteur ordinal (PC) qui pointera alors l'instruction (3).

6. — *FIN* : si l'indicateur Z est à 1, le programme est terminé et l'on va en FIN. Nous avons intérêt à être prévenu d'une telle opération ; or nous savons que le moniteur contient un programme qui affiche — 8085 quand le microprocesseur est prêt à recevoir des ordres ; en lisant le moniteur, nous voyons que ce programme commence en 0008, effectue la sauvegarde des registres avant de commander l'affichage de — 8085. Pour faire «sauter» notre programme en 0008, il suffit d'utiliser l'instruction RST 1 (CF)

Le programme est donc le suivant :

adresse	entrée	instruction			mnémonique	sortie
2000	\$o	01	ab		LXI B, ba	\$o
03		16	00		MVI D 00	
05		79			MOV A, C	
06		82			ADD D	
07		57			DAA	
08		05			MOV D, A	
09		C2	05	20	DCR B	
0C		CF			JNZ \$o RST1	

Nous avons fixé l'adresse de début à programme à 2000, qui est celle de la première case mémoire en RAM sur le kit SDK 85. Cette adresse sera modifiée en fonction de votre micro-ordinateur. Après avoir écrit le programme en mémoire, avec pour exemple a = 05 et b = 04, selon un processus propre à notre machine nous allons le relire pour vérifier qu'il n'y a pas d'erreur.

Exécution du programme

Le programme est lancé par un ordre du type GO 2000 ou GO 2000, 200C. Pour le cas du SDK 85 l'arrivée en 200C nous est signalé par l'affichage de 8085. Le résultat sera obtenu en lisant le contenu du registre D. Nous trouvons 14, c'est-à-dire que $5 \times 4 = 14$! Aurions nous commis une erreur : pour le savoir exécutons le programme en pas à pas.

Exécution en pas à pas

Le travail du microprocesseur est arrêté après l'exécution de chaque instruction ce qui nous permet de le suivre. Ainsi dans le cas du SDK 85 après l'exécution d'une instruction le système affiche l'adresse et le code-opération de l'instruction suivante. En examinant les registres nous obtenons.

Adresse d'arrêt	A	B	C	D	Commentaires
2003	00	04	05	XX	MOV AC a été exécuté ADD D aussi MOV DA aussi DCR B aussi On a sauté en \$o
2005	00	04	05	00	
2006	05	04	05	00	
2007	05	04	05	00	
2008	05	04	05	05	
2009	05	03	05	05	
2005	05	03	05	05	
2006	05	03	05	05	
2007	0A	03	05	05	

Arrêtons-nous un instant : nous savons qu'en hexadécimal, nous comptons de 0 à F, chaque chiffre étant écrit en binaire avec 4 bits, donc A vaut 10 en décimal. Le microprocesseur travaille en binaire et nous lisons le résultat en «code» hexadécimal ; pour lire le résultat en

décimal, il faut utiliser le code DCB («Décimal Codé Binaire») où les bits sont groupés par 4 pour former 9 au maximum. Le 8085 effectue cette opération grâce à l'instruction DAA qui se code 27.

Cette instruction permet d'ajouter 06 au contenu de l'accumulateur si le quartet bas est supérieur à 9 ou si la retenue auxiliaire (notée AC) vaut 1, c'est-à-dire s'il y a eu une retenue qui s'est propagée du quartet *bas* vers le quartet *haut*, d'ajouter 60 si le quartet haut est supérieur à 9 ou si la retenue («Carry») vaut 1 et enfin dans un dernier cas, d'ajouter 66 afin de corriger simultanément les deux quartets.

2000		01 05 04	LXI B, 04 05
03		16 00	MVI D, 00
05	\$o	79	MOV A, C
06		82	ADD D
07		27	DAA
08		57	MOV D, A
09		05	DCR B
0A		C2 05 20	JNZ \$o
0D		CF	RST 1

L'examen des registres en pas à pas donne alors

Adresse	A	B	C	D
2003	00	04	05	XX
05	00	04	05	00
06	05	04	05	00
07	00	04	05	00
08	05	04	05	00
09	05	04	05	05
0A	05	03	05	05
05	05	03	05	05
06	05	03	05	05
07	0A	03	05	05
08	10	03	05	05
09	10	03	05	10
0A	10	02	05	10
05	10	02	05	10
06	05	02	05	10
:	:	:	:	:
0D	20	00	05	20

Pour comprendre l'instruction DAA, refaire le programme en pas à pas avec d'autres valeurs de a et b , toujours inférieures ou égales à 9, par exemple, on aura un $AC = 1$ dans le cas où $a = 8$ car en hexadécimal on a $8 + 8 = 10$.

1	AC
0000	1000
0000	1000
0001	0000

NB1. - On peut interrompre le pas à pas en pressant GO après EXEC, puis EXEC.

NB2. - Nous n'avons considéré que des nombres *positifs*.

CONVERSION:DCB - HEXADECIMAL POUR UN NOMBRE DE DEUX CHIFFRES

But : Etude des masques et des sous-programmes. Préparation à la multiplication de deux nombres de deux chiffres. Initiation au complément à 2.

Principales instructions utilisées : ANI, RLC, CNZ

Dans le programme précédent, si b est un nombre de deux chiffres, nous obtenons un résultat erroné ; mais si le résultat ne comporte pas plus de deux chiffres, par exemple si $a = 5$ et $b = 12$, nous devons trouvé 60 ; or (faites l'expérience), le résultat fourni par le microprocesseur est 90 ! Cela, alors que si $a = 12$ et $b = 5$, le contenu de D est bien 60. Pourquoi ?.

N'oublions jamais que pour notre machine, tous les nombres sont écrits en binaire et que nous les codons pour en faciliter l'écriture et la lecture. Si nous donnons 12 (en décimal) à la machine, elle l'interprète comme la suite de 0 et de 1 0001 0010 ; or, en binaire, ce nombre vaut $16 + 2 = 18$, donc $18 \times 5 = 90$.

Il faudra donc prendre la précaution de traduire le nombre sur lequel est faite la décrementation en hexadécimal.

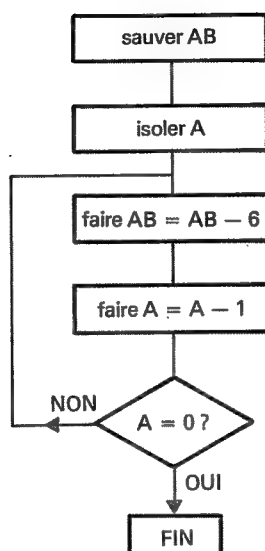
nombre de deux chiffres

Examinons, au préalable, comment s'effectue une soustraction en binaire. Si nous voulons retrancher b du nombre a , nous devons ajouter à a le complément à 2 de b obtenu en remplaçant les 0 par des 1 (et vice versa), puis en ajoutant 1.

— Exemple 1 : faire $5 - 3$

5 s'écrit en binaire (et sur 4 bits)	0101
3 s'écrit dans ce même code	0011
donc -3 se note	1101
ce qui donne $5 - 3$:	0101
(notez le signe + de l'addition	+ 1101
soit :	<u>10010</u>

Organigramme



— Problème : comment isoler A ?

Pour ce faire, nous utilisons la technique du masque. Nous savons que le microprocesseur «sait» faire le «ET» logique dont le tableau est le suivant (pour deux variables) :

Entrées	Sortie
0 0	0
0 1	0
1 0	0
1 1	1

On voit que le «ET» ne donne un 1 *que si les deux bits sont 1*, ou encore, que le résultat est 0 si l'un des bits est à 0. Donc, pour isoler x , nous ferons un «ET» entre le contenu de l'accumulateur (AB) et $F0$ (1111 0000) ; le résultat sera $A0$: nous ne retrouverons aucun des bits constituant B .

Puis, il faut transformer $A0$ en $0A$ et pour cela, nous utiliserons une «rotation», soit à droite, soit à gauche, mais une rotation telle qu'elle recopie aussi le bit de poids fort dans le bit de poids faible ou vice-versa. Dans notre cas, ce sera RLC (ou RRC). L'opération de conversion ne sera à effectuer que si A ne vaut pas zéro ; nous ajouterons donc un test après le masque. Le programme est ainsi le suivant :

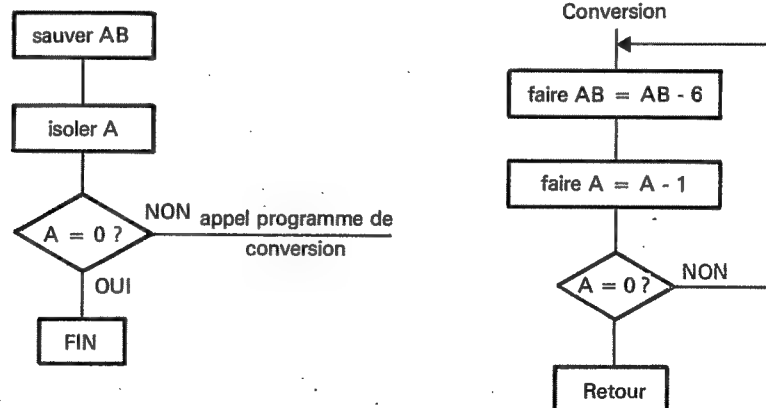
3EAB	MVI	A,OAB	! LE NOMBRE "AB" MIS DANS A
57	MOV	D,A	! "AB" EST SAUVE DANS D
E6F0	ANI	OFO	! "ET" IMMEDIAT, (A) = A0
CA \$1:	JZ	\$1	! SI A0=0 PAS DE CONVERSION
07	RLC		
07	RLC		
07	RLC		
07	RLC		! (A)=0A
47	MOV	B,A	! (B)=(A)=0A
0EFA	MVI	C,OFA	! (C)=FA=-6

! (R) = CONTENU DU REGISTRE R

On a ici écrit le même programme que précédemment, sans l'instruction MVI D, 00 puisqu'il faut que D contienne AB. Le branchement \$1 est placé à la fin. Le résultat de AB converti en hexadécimal est lu dans D.

Sous-programme

Ce programme, très court, ne justifie pas en soi l'utilisation d'un sous-programme mais va nous permettre d'étudier le fonctionnement d'une telle opération. Pour ce faire, nous écrivons l'organigramme comme suit



Pour programmer avec sous-programme (s), il faut absolument initialiser le pointeur de pile (SP). Ce registre pointe le haut d'une pile qui se remplit de haut en bas (adresse haute en premier) ; au moment d'un appel de sous-programme, l'adresse à laquelle on doit revenir à la fin du sous-programme est mémorisée dans la pile et le contenu du registre SP est diminué de 2. Lors du retour, cette adresse est mise dans le compteur ordinal et le contenu du registre SP est augmenté de 2. Tout ceci pourra être observé grâce au fonctionnement en pas à pas et à l'examen des registres et de quelques cases mémoires.

Pour appeler un sous-programme, on écrit un appel (CALL) qui peut être conditionné ; le sous-programme se termine par un retour (RETURN) qui peut également être conditionné. Voici le problème de conversion ainsi traité :

```

1          .TITLE CONVER
2
3          ;CONVERSION DCB-HEXA POUR UN NOMBRE
4          ; DE DEUX CHIFFRES
5
6 0000      . =02000
7
8 2000 31C020    LXI    SP,020C0    ;INITIALISATION DE SP
9 2003 3EAB"     MVI    A,0AB      ; (A) = "AB"
10 2005 57       MOV    D,A        ;ET SAUVE DANS D
11 2006 E6F0     ANI    0F0        ;"ET" IMMEDIAT, (A)=A0
12 2008 C40C20   CNZ    CONV       ;SI A0=0 PAS DE CONV.
13 200B CF       RST     1         ;FIN ,LE RESULTAT EST
14             ;LU DANS D
15             ;
16 200C 07      CONV:  RLC
17 200D 07      RLC
18 200E 07      RLC
19 200F 07      RLC
  
```

20	2010	47	MOV	B,A	%B CONTIENT "0A"
21	2011	7A	MOV	A,D	%A CONTIENT "AB"
22	2012	C6FA	ADI	0FA	%ON RETRANCHE 6 A "AB"
23	2014	05	DCR	B	%
24	2015	C21220	JNZ	%0	%SI (B)=0 ON REVIENT ,
25					%SINON ON BOUCLE
26	2018	57	MOV	D,A	% (D) = RESULTAT
27	2019	C9	RET		
28					
29		0000	.END		

Exécution en pas à pas

Soit à exécuter le programme en pas à pas jusqu'à 2008. On va examiner les registres :

A	contient	A0
D	contient	AB
SPH	contient	20
SPL	contient	C0

Examinons les cases mémoires :

20 BE	nn	} Valeurs aléatoires
20 BF	nn	
20 C0	nn	

On reprendra ensuite ce pas à pas ; le microprocesseur, si A est différent de zéro, saute en 2010. Examinons les registres :

A	contient	A0
D	contient	AB
SPH	contient	20
SPL	contient	BE

Examinons les cases mémoires :

20 BE	0B	} Adresse à laquelle on doit revenir
20 BF	20	
20 C0	nn	

Reprenons le pas à pas ; au bout d'un certain nombre de boucles, le microprocesseur saute en 200B. Examinons les registres :

A	contient	AB	converti
D	contient	AB	
SPH	contient	20	
SPL	contient	C0	

MULTIPLICATION DE DEUX NOMBRES DE DEUX CHIFFRES

But : Synthèse des programmes précédents.

Principale instruction utilisée : CMP.

Le seul problème supplémentaire que nous rencontrons est de prévoir que le résultat ait 4 chiffres. Dans ce cas, il faudra utiliser la paire D à la place du registre D ; le registre E contiendra les dizaines et les unités, le registre D les centaines et milliers.

Au cours de l'addition, nous sommes prévenus du dépassement de capacité de l'accumulateur par l'indicateur de retenue «carry» (CY) qui passe à 1. A chaque fois qu'il y aura dépassement, nous incrémenterons de 1 le contenu du registre D *sans oublier* de le convertir *immédiatement* en décimal. Toutes ces considérations nous conduisent au programme suivant : nous traitons la conversion et les centaines comme des sous-programmes ; nous avons ajouté quelques instructions qui permettent de mettre dans B le plus petit nombre pour minimiser la durée d'exécution. Pour ce faire, on utilise l'instruction CMPr. Celle-ci compare les contenus de A et d'un registre *r* sans les détruire et nous renseigne sur leur valeur relative en affectant les indicateurs comme suit :

Relation	Z	CY
(A) < (r)	0	1
(A) = (r)	1	0
(A) > (r)	0	0

Donc, si seul le caractère «plus petit que» nous intéresse, on teste le «carry» ; si seul le caractère «égal à» nous intéresse, on teste le «flag» Z (l'indicateur Z) ; mais s'il faut faire un tri : *plus grand, égal, plus petit*, on peut commencer par tester Z puis CY selon la séquence suivante :

- CMP r
- JZ «égal»
- JC «plus petit»
- et début du programme pour «plus grand».

On suppose, ici, que les nombres ne sont pas nuls.

```

1          .TITLE  MULDEC
2
3          ;MULTIPLICATION DECIMALE DE DEUX
4          ;NOMBRES DE DEUX CHIFFRES
5
6 0000          . =      02000
7
8 2000 31C020    LXI      SP,020C0    ;INITIALISATION DE SP
9 2003 110000    LXI      D,00000    ;INITIALISATION DE DE
10 2006 01CDAB"  LXI      B,0ABCD    ;ENTREE DES 2 NOMBRES
11 2009 79        MOV      A,C        ;A CONTIENT "CD"
12 200A B8        CMP      B          ;
13 200B D21020    JNC      $0          ;SI CY=0 "CD">"AB" ON
14              ;NE PERMUTE PAS SINON...
15 200E 48        MOV      C,B        ;"AB" EST MIS DANS C
16 200F 47        MOV      B,A        ;"CD" EST MIS DANS B
17 2010 78        MOV      A,B        ;(A)=LE PLUS PETIT
18 2011 B7        ORA      A          ;TEST (B)=(A)=0 ?
19 2012 CA2520    JZ       FIN        ;
20 2015 E6F0      ANI      OF0        ;TEST DES DIZAINES
21 2017 C42C20    CNZ      CONV      ;SI DIZ. NON NULLES
22              ;APPEL DU S.P.CONV.
23 201A 79        MOV      A,C        ;(A)=LE PLUS GRAND
24 201B 83        ADD      E          ;
25 201C 27        DAA          ;AJUSTEMENT DECIMAL
26 201D 5F        MOV      E,A        ;(E)=RESULTAT
27 201E IC2620    CC       CENT      ;SI RETENUE APPEL "CENT"
28 2021 05        DCR      B          ;
29 2022 C21A20    JNZ      $1        ;SI (B)=0 FIN
30 2025 CF        FIN:    RST      1    ;FIN
31
32          ;CENT. INCREMENTE LE NOMBRE DE CENTAINES
33
34 2026 B7        CENT:    ORA      A          ;ON MET CY A ZERO
35 2027 14        INR      D          ;INCREMENTATION
36 2028 7A        MOV      A,D        ;DES CENTAINES
37 2029 27        DAA          ;
38 202A 57        MOV      D,A        ;CENTAINES DANS D
39 202B C9        RET          ;
40
41          ;CONV. CONVERTIT LE NOMBRE CONTENU
42          ;DANS B EN HEXADECIMAL
43
44 202C 07        CONV:    RLC          ;
45 202D 07        RLC          ;
46 202E 07        RLC          ;
47 202F 07        RLC          ;
48 2030 5F        MOV      E,A        ;E=MEMOIRE TAMPON
49 2031 78        MOV      A,B        ;
50 2032 C6FA      $2:    ADI      OFA    ;
51 2034 1D        DCR      E          ;
52 2035 C23220    JNZ      $2        ;
53 2038 47        MOV      B,A        ;
54 2039 C9        RET          ;

```

4

AFFICHAGE

But : Utilisation des sous-programmes du moniteur et précautions à prendre - Utilisation du 8279.

Principales et nouvelles instructions : PUSH, POP, STA.

Il est évident que la recherche du résultat n'est pas très souple ; il est préférable de pouvoir le voir sur les afficheurs à 7 segments. Il existe deux programmes d'affichage dans le moniteur. L'un affiche le contenu de l'accumulateur en «données», l'autre le contenu de la paire D en «adresses». Ces programmes utilisent les registres A, B, C, D, E, H, L et détruisent par conséquent ce qui s'y trouvait antérieurement ; donc, il faut prendre des précautions lors de leur emploi. Ces précautions consistent à sauvegarder ces contenus en utilisant les instructions PUSH avant appel du sous-programme, et POP après :

— PUSH *p* sauve une paire de registres B, D, H, A plus les «flags» (PSW) en rangeant la partie haute à l'adresse pointée par le contenu du registre SP décrémenté de un et la partie basse à l'adresse pointée par le contenu au registre SP diminué de deux (tout comme pour l'instruction CALL).

— POP opère de façon inverse, le contenu du registre SP retrouvant sa valeur initiale.
Attention à l'ordre ! A moins de vouloir charger une paire avec le contenu de l'autre : exemple, le test de AC dans la soustraction (à ce sujet voir le programme 18).

En résumé : pour afficher le contenu de A, on appelle le programme UPDDT qui commence en 036E ; pour afficher le contenu de DE, on appelle le programme UPDAD qui commence en 0363. Dans le cas du programme précédent, se terminant par un RST 1, le résultat est dans DE ; pour l'afficher, il suffit de remplacer RST 1 par :

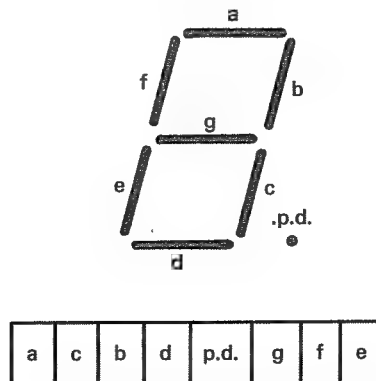
CD 63 03 CALL UPDDT

76 HLT

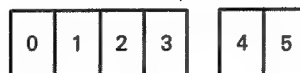
On termine ici par HLT : le microprocesseur est alors complètement arrêté (sauf l'horloge) ; si nous avons remis RST 1, il y aurait eu affichage bref du résultat puis un retour à «8085». Si l'on veut un point à droite du dernier chiffre, il faut faire précéder l'appel du sous-programme de MVI B, 01 ; sinon, il faut mettre MVI B, 00 et donc charger B avec zéro ; en cas d'oubli, on obtiendra ou non un point...

LE CIRCUIT D'INTERFACE 8279

Au lieu d'utiliser les sous-programmes du moniteur, on peut écrire un programme plus adapté à nos besoins. Pour ce faire, il faut connaître le fonctionnement du circuit 8279 qui gère les afficheurs et le clavier. Chaque afficheurs comporte 7 segments et un «point» décimal ; le caractère à afficher peut donc être codé sur 8 bits. Les segments étant repérés par les lettres *a, b,...* à *g*, le câblage du kit nous donne la correspondance suivante entre les segments et les bits :



Un bit à zéro allume le segment correspondant ; ainsi, 4 à pour code 10011001 = 99 H. Pour faire apparaître un caractère il faut préciser au circuit 8279 la position de ce caractère :



et lui «dire» si le caractère suivant éventuel sera mis à côté ou à la place de ce caractère. *Il faut donc lui envoyer un mot d'ordre* (CW = «Command word»). Ce mot est stocké dans un registre propre du 8279 qui a pour adresse 19XX (XX = indéterminés). Les caractères à afficher sont stockés dans les cases d'une mémoire dont l'adresse est 18XX : il suffit d'une adresse, le mot de commande précisant si le caractère envoyé remplace ou suit (case mémoire suivante) le précédent.

Mot de commande

Si on désire que le caractère envoyé remplace le précédent, le mot de commande est 8Y où Y (0 à 5) est la position de l'afficheur. Si on désire que le caractère envoyé se place à côté (à droite) du précédent le mot est 9Y, où Y est la position du premier caractère. Ainsi :

— Exemple 1 : soit le programme suivant

```

1          .TITLE AFF-1
2
3          ;COMMANDE AFFICHAGE DU 8279 (1)
4
5 0000          .*=02000
6
7 2000 3E92      MVI      A,092          ;92H = MOT DE COMMANDE
8 2002 320019    STA      01900          ;EST RANGE EN 1900
9 2005 3E3A      MVI      A,03A
10 2007 320018   STA      01800
11 200A 3E3E      MVI      A,03E
12 200C 320018   STA      01800
13 200F 76       HLT

```

fait «écrire» :



le E étant toujours présent, il signifie que le 8085 travaille, mais il peut être remplacé par un signe si le programme le demande.

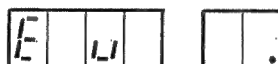
— Exemple 2 : soit le programme suivant

```

1          .TITLE AFF-2
2
3          ;COMMANDE AFFICHAGE DU 8279 (2)
4
5 0000          .*=02000
6
7 2000 3E82      MVI      A,082
8 2002 320019    STA      01900
9 2005 3E3A      MVI      A,03A
10 2007 320018   STA      01800
11 200A 3E3E      MVI      A,03E
12 200C 320018   STA      01800
13 200F 76       HLT

```

fait «écrire» :



le **I** ayant remplacé le **I**

N.B.— Si l'on désire gagner de la place, on remplace les STA qui occupent trois octets par des MVI M qui n'en occupent que deux et permettent la suppression de MVI A, ce qui donne : pour l'exemple 1 :

1			.TITLE AFF-3	
2				
3			!COMMANDE AFFICHAGE 8279 (3)	
4				
5	0000		.=02000	
6				
7	2000 210019	LXI	H,01900	!INITIALISATION DE HL
8	2003 3692	MVI	M,092	!CASE MEMOIRE POINTEE
9				!PAR (HL) CONTIENT LE
10				!MOT DE COMMANDE (C.W)
11	2005 25	DCR	H	! (H) =18
12	2006 363A	MVI	M,03A	
13	200B 363E	MVI	M,03E	
14	200A 76	HLT		

Ici, on a commencé par charger dans la paire H, L l'adresse de la cellule mémoire 1900 ; il suffira d'utiliser ensuite les instructions qui, par leur définition imposée par le constructeur, se réfèrent à (H,L) où elles trouveront l'adresse utile. Nous avons ainsi gagné 5 octets !
Le circuit d'interface 8279 offre d'autres possibilités que nous étudierons ultérieurement.

ENTREE DE DONNEES AU CLAVIER

But : Gestion des interruptions.

Principales ou nouvelles instructions utilisées : SIM, EI.

Le circuit 8279 gère également le clavier. Lorsqu'une touche est pressée, sa valeur codée est stockée dans une mémoire interne au 8279 qui comporte 8 cases. Dès qu'une case est pleine et tant qu'elle n'a pas été lue (vidée) par le microprocesseur, l'une des pattes du circuit qui est normalement à 0V est portée à 5 V. Le kit est construit de façon que le 8085 soit prévenu qu'une touche a été pressée ; il pourra en tenir compte ou l'ignorer ; la patte du 8279 est en effet reliée à une entrée « interruption » du 8085.

Interruption

Dans la majorité des cas, un micro-ordinateur est utilisé pour contrôler divers processus. Il peut y avoir des « alarmes » (température, fuite...) ; si le programme est conçu pour inspecter chaque circuit et attendre que l'événement voulu se produise, il occupera l'essentiel de son temps à...ne rien faire. Il est plus judicieux de lui faire effectuer une ou plusieurs tâches utiles qui seront *interrompues* lors de l'apparition d'un événement extérieur qui sera traité, avant de revenir au programme principal. Il faut donc que l'interruption soit *prévue et autorisée*.

Une interruption est gérée par un *sous programme* qui est *appelé* automatiquement dès qu'elle est *reconnue* ; cela veut dire qu'il faut *initialiser* le pointeur de pile (SP), écrire les programmes d'interruption comme des sous-programmes (sauvegarde éventuelle des registres, puis restitution et retour).

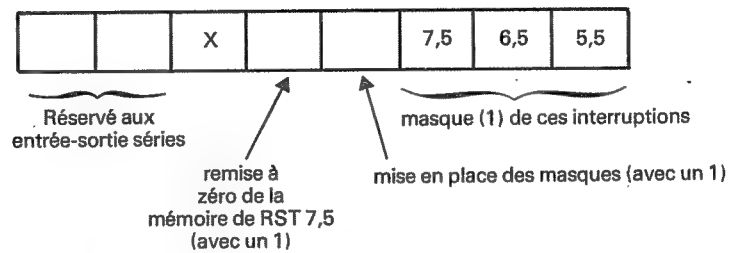
Dans le 8085, les interruptions peuvent être de deux types : *vectorisées*, ou non. Une interruption *vectorisée* est une interruption dont l'adresse du début du programme est définie par le constructeur ; dans le cas contraire, cette adresse est définie par le programmeur et il est nécessaire d'utiliser un ou plusieurs circuits périphériques.

Le microprocesseur 8085 est doté de quatre types d'interruptions vectorisées : TRAP, RST 7.5, RST 6.5, RST 5.5 dont les programmes commencent en 0024, 003C, 0034, 002C. Elles sont classées par ordre de priorité décroissante (RST 5.5 ne peut interrompre RST 7.5 mais l'inverse peut se faire). Il existe également une entrée INTR qui nécessite un circuit donnant l'adresse du programme (ce qui est décrit en détail dans un ouvrage édité par Intel : *Péripal Handbook* : 8212 - 8259). Après un « reset », les interruptions sont *masquées* (sauf TRAP qui ne peut l'être) ; donc, si on veut les utiliser il faudra :

- 1 - les démasquer ;
- 2 - les autoriser.

Les opérations s'effectuent avec les deux instructions suivantes :

— *SIM (Set Interrupt Mask)* : il faut que l'accumulateur soit chargé avec un mot dont les bits ont les fonctions ci-dessous :



— *EI (Enable Interrupt)* qui permet au microprocesseur de gérer une interruption non masquée. Il faut, dans le programme de gestion d'une interruption (au début, ou à la fin, ou en cours), les autoriser à nouveau à chaque fois qu'on veut qu'elles puissent intervenir.

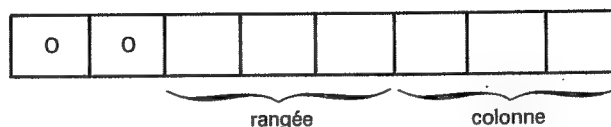
Cas du 8279. Dans le kit, la sortie du 8279, qui passe à 5 V quand une touche au moins est pressée, est reliée à l'entrée de l'interruption RST 5.5. Donc, si nous voulons que le 8085 enregistre l'événement : «touche pressée», il faut autoriser l'interruption RST 5.5 selon la séquence.

```
MVI A, 08 (ou 0C, ou 0E)
SIM
EI
```

Dans ce cas, si une touche est pressée, un court programme moniteur range le code de la touche en 20FE ; un autre programme (RDKBD : «*Read Keyboard*») commençant en 02E7 attend que le contenu de 20FE corresponde au code d'une touche et le range dans l'accumulateur.

Code d'une touche

Le mot correspondant à une touche est le suivant :



Le clavier est une matrice à 8 colonnes et 8 rangées (au maximum). En l'absence de touche pressée, 20FE contient 80 (ce qui ressort du programme moniteur).

Programme d'entrée «d'une touche» avec affichage

Le programme suivant nous fournira le code de chaque touche au clavier, à l'exception évidente de «RESET» et «VECT.INT».

```
1 .TITLE CLAV-1
2
3 ;ENTREE D'UNE TOUCHE ET AFFICHAGE
4
5 0000 . =02000
6
7 02E7 RDKBD =002E7
8 036E UPDDT =0036E
9
```

```

10          .MACRO  SIM
11          .BYTE   030
12          .ENDM
13
14          ;ASSEMBLEUR 8080 NE CONNAIT PAS "SIM"
15          ;ON "CREE" L'INSTRUCTION PAR UNE "MACRO"
16
17 2000 31C020          LXI      SP,020C0          ;IL FAUT INITIALISER SP
18
19                                     ;CAR UNE INTERRUPTION
20                                     ;EST GEREE PAR UN
21                                     ;SOUS-PROGRAMME
22 2003 3E08          MVI      A,008          ;AUTORISATION
23 2005          SIM                                     ;DE
24 2006 FB          $0:      EI          ;RST 5.5
25 2007 CDE702          CALL    RDKBD
26 200A CD6E03          CALL    UPDDT          ;AFFICHAGE DE (A)
27 200D C30620          JMP     $0

```

Tant qu'une touche n'est pas pressée on voit

E

Si on presse A nous avons

E A

etc.

N.B. (1)

Ce code est à noter pour comprendre des programmes ultérieurs.

N.B. (2)

On ne peut effectuer ce programme en pas à pas car on tournerait dans RDKBD, à moins de mettre en 20FE (par SUBST.MEM) un mot dont le bit le plus à gauche est 0. Dans le cas d'une interruption, on met un point d'arrêt («break-point») au début du sous-programme d'interruption ; ce point d'arrêt est obtenu par...RST 1 (CF). Au moment où «— 8085» s'affiche, on continue en pas à pas si besoin est.

ADDITION EN HEXADECIMAL AVEC AFFICHAGE DU 1^{er} OPERANDE OU DU RESULTAT

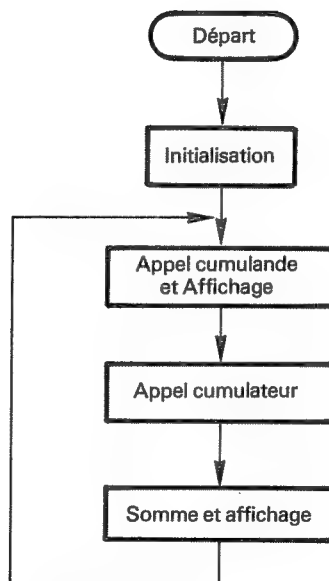
But : Prouver que toutes les séquences à exécuter doivent être prévues par le programmeur car le microprocesseur ne sait pas réparer un oubli et n'a aucune imagination propre. Principales et nouvelles instructions utilisées : LDA, STA.

Cet exercice découle directement du précédent et est à peine plus compliqué. Il va cependant constituer une étape de plus. En effet, après affichage de la touche pressée - premier opérande - on presse une deuxième touche dont le code est additionné à celui de la première, l'affichage donnera aussitôt le résultat. Puis, on peut recommencer une autre addition, toujours en hexadécimal avec un digit par opérande.

Une difficulté surgit : le sous-programme du moniteur qui commande l'affichage, en se déroulant, détruit le contenu des registres (ce que nous apprend ce programme, proposé dans le manuel *Intel* qui accompagne le SDK-85). Par conséquent, il va falloir «sauvegarder» le premier opérande ailleurs que dans les registres, c'est-à-dire en mémoire RAM. Très arbitrairement, l'auteur a choisi la cellule d'adresse 20B0 pour recevoir le cumulande.

Le programme se déroule de la façon suivante :

- Les cinq premières instructions servent à l'initialisation : pointeur de pile, masque d'interruptions et autorisations de celles-ci.
- En 200A : on a frappé une valeur pour le cumulande, et elle se trouve dans l'accumulateur. On va la transférer dans la cellule de sauvegarde d'adresse 20B0, en mémoire. Le code instruction, lui, s'écrira 32 pour l'ordre de rangement («STA», pour «Store Accumulator»), suivi de l'adresse avec octets inversés : B020 (au lieu de 20B0), comme on l'a déjà fait précédemment.
- 200D : on commande l'affichage.



- 2010 : après ré-autorisation des interruptions, on appelle le cumulateur (instruction 2011) qui vient se loger en (A) dès qu'une touche est enfoncée.
- 2014 : à nouveau, on va sauvegarder (A) en le rangeant, mais dans le registre B cette fois.
- 2015 : puis, on rappelle le cumulande dans A. On notera que dans le mnémonique, c'est toujours le destinataire qui est cité le premier, ici une adresse mémoire.

	6	; ADDITION AVEC ENTREE CLAVIER ET AFFICHAGE			
	7	; (1 CHIFFRE)			
	8				
02E7	9	RDKBD	EQU	02E7H	
036E	10	UPDDT	EQU	036EH	
	11				
2000	12	ORG	2000H		
	13				
2000 31C020	14	DEBUT:	LXI	SP, 20C0H	
2003 3E08	15		MVI	A, 08H	; COMME
2005 30	16		SIM		; LE
2006 FB	17	BOUCLE:	EI		; PROGRAMME
2007 CDE702	18		CALL	RDKBD	; PRECEDENT
200A 32B020	19		STA	20B0H	; RANGEMENT EN MEMOIRE
200D CD6E03	20		CALL	UPDDT	; AFFICHAGE 1ER OPERANDE
2010 FB	21		EI		; RE-AUTOR. INTER.
2011 CDE702	22		CALL	RDKBD	; APPEL 2EME OPERANDE
2014 47	23		MOV	B, A	
2015 3AB020	24		LDA	20B0H	; RAPPEL 1ER OPERANDE
2018 80	25		ADD	B	; ADDITION
2019 CD6E03	26		CALL	UPDDT	; AFFICHAGE
201C C30620	27		JMP	BOUCLE	; BOUCLE
	28				
2000	29	END	DEBUT		

- 2018 : cela fait, on peut additionner (B) à, implicitement, le contenu de A,
- 2019 : puis afficher le résultat, et recommencer si on le désire (201C).

Si on ne voulait exécuter qu'une seule addition, il suffirait de remplacer la dernière instruction, à l'adresse 201C, par un ordre HLT, qui se code 76. On ne pourrait recommencer qu'en faisant RESET, etc.

ADDITION AVEC AFFICHAGE SIMULTANE DU CUMULANDE ET DE LA SOMME

But - Introduire l'usage d'un autre sous-programme d'affichage contenu dans le moniteur.

Encore un petit pas : cette fois, on va afficher à la fois le cumulan, dans le champ «Données» de l'afficheur, et la somme, dans le champ «Adresses». Pour cela, il suffit de modifier l'adresse d'affichage et, au lieu de passer par le sous-programme UPDDT du moniteur, d'emprunter celui labellé UPDAD.

Les douze premières instructions ne changent pas. Mais ensuite :

— Adresse 2019 : la somme, qui se trouve dans l'accumulateur, est transférée dans le registre E. C'est le contenu de ce registre qu'utilise le sous-programme du moniteur appelé UPDAD. Ainsi en ont décidé ses auteurs !

— 201A : on appelle le sous-programme d'affichage dans le champ «Adresses», UPDAD, domicilié à partir de l'adresse 0363 dans le moniteur, ce qui devient bien sûr 6303 dans le code numérique.

Puis, on peut recommencer à volonté. Il n'y a pas là la moindre difficulté, sauf qu'on traîne toujours cet hexadécimal dont l'interprétation n'est pas immédiate.

	1	\$MOD85	
	2		
	3		
	4	NAME	ADDAFF01
	5		
	6	; ADDITION AVEC ENTREE CLAVIER ET AFFICHAGE	
	7	; (1 CHIFFRE)	
	8		
02E7	9	RDKBD	EQU 02E7H
036E	10	UPDDT	EQU 036EH
0363	11	UPDAD	EQU 0363H
	12		
2000	13	ORG	2000H
	14		
2000 31C020	15	DEBUT:	LXI SP, 20C0H
2003 3E08	16	MVI	A, 08H ; COMME
2005 30	17	SIM	;
2006 FB	18	BOUCLE:	EI ; LE
2007 CDE702	19	CALL	RDKBD ;
200A 32B020	20	STA	20B0H ; PROGRAMME
200D CD6E03	21	CALL	UPDDT ;
2010 FB	22	EI	; PRECEDENT
2011 CDE702	23	CALL	RDKBD
2014 47	24	MOV	B, A
2015 3AB020	25	LDA	20B0H
2018 80	26	ADD	B
2019 5F	27	MOV	E, A ; CHARGE BUFFER AFFICHAGE
201A CD6303	28	CALL	UPDAD ; AFFICHE ZONE "ADRESSES"
201D C30620	29	JMP	BOUCLE ; BOUCLE

ADDITION EN DECIMAL DE DEUX FOIS DEUX DIGITS, AVEC RESULTAT AFFICHE SUR TROIS DIGITS

But : Introduire le traitement par quartet.

Principales et nouvelles instructions utilisées : RLC, RAL.

Les additions précédentes restent très élémentaires. Cet exercice va montrer comment on peut accroître le nombre de digits sur lesquels porte cette opération.

Le cumulande, tout comme le cumulateur, comportent deux digits. Chaque digit correspond à un quartet (4 bits binaires) ; il faut donc distinguer le *quartet de faible poids* du *quartet de fort poids* pour former l'octet. Le résultat sera sur trois digits significatifs ; par conséquent, il faudra aussi se préoccuper de la propagation d'une retenue éventuelle entre le premier octet (de faible poids) et le second octet (de fort poids). Voici ce programme :

— Les 5 premières instructions sont immuables ! Elles vont jusqu'à l'introduction du quartet de fort poids dans l'accumulateur, pour le cumulande. Le sous-programme RDKBD ne sachant pas distinguer un quartet de faible poids d'un quartet de fort poids, *il loge tout dans le quartet de faible poids de l'octet rangé dans l'accumulateur.*

Par exemple, si l'on veut former 90 en décimal, soit 1001 0000 en BCD (en décimal codé binaire), avec 1001 pour le quartet de fort poids et 0000 pour celui de faible poids, on créera tout d'abord le 9, soit 1001, en frappant la touche 9 après appel de RDKBD. Le sous-programme rangera 9 dans l'accumulateur sous la forme : 0000 1001. Il faut donc déplacer de 4 rangs vers la gauche cet octet pour retrouver 1001 0000. C'est l'objet des quatre instructions suivantes :

ISIS-II 8080/8085 MACRO ASSEMBLER, V4.0 ADDAFF PAGE 1

LOC	OBJ	LINE	SOURCE STATEMENT
		1	\$MOD85
		2	
		3	
		4	
		5	NAME ADDAFF02
		6	
		7	; ADDITION DECIMALE AVEC ENTREE CLAVIER
		8	; ET AFFICHAGE (2 CHIFFRES)
		9	
02E7		10	RDKBD EQU 02E7H
036E		11	UPDDT EQU 036EH
0363		12	UPDAD EQU 0363H
		13	
2000		14	ORG 2000H
		15	

2000 31C020	16	DEBUT:	LXI	SP, 20COH;	COMME
2003 3E08	17		MVI	A, 08H	;
2005 30	18		SIM		;
2006 FB	19	BOUCLE:	EI		;
2007 CDE702	20		CALL	RDKBD	;
200A 07	21		RLC		;
200B 07	22		RLC		;
200C 07	23		RLC		;
200D 07	24		RLC		;
200E 47	25		MOV	B, A	;
200F FB	26		EI		;
2010 CDE702	27		CALL	RDKBD	;
2013 80	28		ADD	B	;
2014 5F	29		MOV	E, A	;
2015 FB	30		EI		;
2016 CDE702	31		CALL	RDKBD	;
2019 07	32		RLC		;
201A 07	33		RLC		;
201B 07	34		RLC		;
201C 07	35		RLC		;
201D 47	36		MOV	B, A	;
201E FB	37		EI		;
201F CDE702	38		CALL	RDKBD	;
2022 80	39		ADD	B	;
2023 83	40		ADD	E	;
2024 27	41		DAA		;
2025 5F	42		MOV	E, A	;
2026 3E00	43		MVI	A, 00H	;
2028 17	44		RAL		;
2029 57	45		MOV	D, A	;
202A CD6303	46		CALL	UPDAD	;
202D C30620	47		JMP	BOUCLE	;

— Adresses 200A à 200D : chacune de ces 4 instructions décale d'un cran vers la gauche le contenu de l'accumulateur. RLC vient de «Rotate Left», c'est-à-dire : rotation vers la gauche. On a donc positionné le quartet de fort poids.

— 200E : on va ranger provisoirement l'octet ainsi formé dans le registre B.

— Après avoir ré-autorisé les interruptions, on appelle à nouveau RDKBD, on frappe sur une autre touche et on obtient le second digit du cumulateur, qui vient lui aussi se ranger dans l'accumulateur où il prend directement sa place : celle du quartet de faible poids.

On notera que l'on n'a pas effacé le contenu précédent de l'accumulateur. Comme dans toutes les opérations de mouvement de données, un nouveau chargement efface l'ancien. C'est pourquoi on passe par l'opération de sauvegarde dans le registre B.

— 2013 : il reste à additionner l'octet de (B), qui donne le quartet de fort poids (le reste est constitué par des zéros) et le contenu de l'accumulateur.

— 2014 : on range le résultat dans le registre E.

— De 2015 à 2022, on répète rigoureusement le même processus pour obtenir le cumulateur.

— 2023 : celui-ci étant dans l'accumulateur, on exécute l'addition avec le cumulateur, contenu dans B,

— 2024 : et l'on effectue la correction décimale.

- 2025 : le résultat est rangé dans le buffer dont le contenu sert au sous-programme d'affichage. On dispose par conséquent de la somme sur deux digits. Mais le troisième ?
- 2026 : s'il y a un troisième digit, c'est un 1 qui résulte d'une retenue provenant de l'addition du cumulande au cumulateur. Cette retenue, c'est le bit *indicateur de retenue*, ou *Carry* (CY en abrégé) qui l'a mise en mémoire.
Vérifions-le, en remettant ici l'accumulateur à zéro, puis :
- 2028 : en décalant le contenu du bit de Carry dans l'accumulateur par décalage à gauche. L'ordre RAL provient de «Rotate A Left Through Carry», pour *rotation du contenu de l'accumulateur à gauche via le bit de carry*. Le contenu de ce dernier va passer à la place du bit de plus faible poids de l'accumulateur.
- 2029 : que ce soit un zéro ou un 1, on va le transférer dans le buffer du second octet d'affichage, soit D, qu'utilise le sous-programme UPDAD visant la zone «Adresses» de l'afficheur.
- On appelle UPDAD, et si l'on veut on recommence.

Introduisez ce programme et exécutez-le. Il est un peu plus long et vous risquez de commettre des erreurs de frappe en le composant. N'hésitez pas à vérifier ce que vous avez entré, à rappeler des adresses pour contrôler leur contenu, jusqu'à ce qu'il «tourne» comme prévu. Attention : les touches hexadécimales sont à nouveau interdites pour composer les opérandes à additionner !



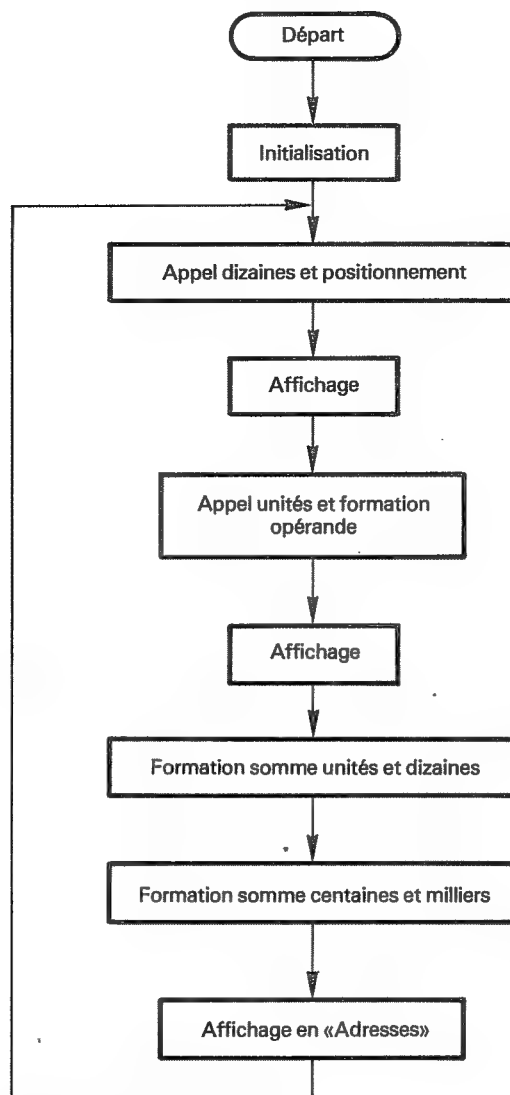
L'INVENTAIRE

*But : Montrer l'utilisation de cellules mémoires comme zones de stockage temporaire.
Principales instructions utilisées : STA ; LDA ; ACI.*

Comme pour un inventaire, on va créer ici des nombres décimaux de deux chiffres, les afficher à la droite de l'afficheur (zone «Données»), et totaliser ceux-ci au fur et à mesure de leur création. La somme est affichée sur 4 digits dans la zone «Adresses». Ainsi et en permanence, on lit, et le total cumulé, et le dernier opérande introduit.

A nouveau, ce programme dérive des précédents, avec quelques nouvelles instructions. D'autre part, on s'est réservé l'usage de trois cellules mémoires choisies assez arbitrairement parmi celles qui restent disponibles et consacrées à :

- Adresse 20B0 : stockage de l'opérande, au fur et à mesure de sa formation. On frappe d'abord le chiffre des dizaines, puis celui des unités. Le total occupe un octet.
- Adresse 20B1 : stockage du total cumulé pour les unités et les dizaines.
- Adresse 20B2 : stockage du total cumulé pour les centaines et les milliers.



En effet, on ne peut stocker ces valeurs dans les registres internes du CPU, dont le contenu est détruit par le sous-programme d'affichage.

Détaillons ce programme, en insistant sur les nouvelles instructions :

— L'initialisation, qui va des instructions d'adresse 2000 à 200E, demande deux instructions supplémentaires, en 2005 et 2008, pour la remise à zéro des adresses des cellules mémoires qui stockeront les totaux. Pour cela, on transfère dans ces adresses le contenu de l'accumulateur, lui-même mis à zéro.

— 200F : on appelle le chiffre des dizaines de l'opérande en cours, et on frappe une touche décimale du clavier. Comme dans le programme précédent, on exécute quatre décalages pour le positionner à gauche de l'octet contenu dans l'accumulateur. On range provisoirement ce digit en mémoire, à l'adresse 20B0, et l'on appelle le sous-programme d'affichage (instruction 2019).

— 201C : on reprend le même processus pour le chiffre des unités, rangé provisoirement dans B, puis rappelé pour être ajouté à celui des dizaines (rappelé dans A), afin de former l'octet complet : le quartet de poids fort représente les dizaines, celui du poids faible les unités. Le résultat est préservé, toujours en 20B0, et affiché.

ISIS-II 8080/8085 MACRO ASSEMBLER, V4.0

INVENT PAGE 1

LOC	OBJ	LINE	SOURCE STATEMENT
		1	\$MOD85
		2	
		3	
		4	NAME INVENTAIRE
		5	
		6	; ADDITION AVEC ENTREE CLAVIER ET AFFICHAGE
		7	; (4 CHIFFRES)
		8	
02E7		9	RDKBD EQU 02E7H
036E		10	UPDDT EQU 036EH
0363		11	UPDAD EQU 0363H
		12	
2000		13	ORG 2000H
		14	
2000 31C020		15	DEBUT: LXI SP, 20C0H
2003 3E00		16	MVI A, 00H
2005 32B120		17	STA 20B1H ; R.A.Z. DIZ. ET UNITES
2008 32B220		18	STA 20B2H ; R.A.Z. MIL. ET CENT.
200B 3E08		19	MVI A, 08H ;
200D 30		20	SIM ;
200E FB		21	BOUCLE: EI ;
200F CDE702		22	CALL RDKBD ; VOIR
2012 07		23	RLC ;
2013 07		24	RLC ; PROGRAMME
2014 07		25	RLC ;
2015 07		26	RLC ; PRECEDENT
2016 32B020		27	STA 20B0H ; RANGEMENT PROVISoire
		28	; DES DIZAINES
2019 CD6E03		29	CALL UPDDT ; AFFICHAGE
201C FB		30	EI
201D CDE702		31	CALL RDKBD ; CHIFFRE DES UNITES
2020 47		32	MOV B, A

2021	3AB020	33	LDA	20B0H	
2024	80	34	ADD	B	;FORMATION
2025	32B020	35	STA	20B0H	;ET RANGEMENT OPERANDE
2028	CD6E03	36	CALL	UPDDT	;ET AFFICHAGE
202B	3AB020	37	LDA	20B0H	;RAPPEL OPERANDE
202E	47	38	MOV	B, A	;ET RANGEMENT DANS B
202F	3AB120	39	LDA	20B1H	;APPEL DIZ. ET UNITES
		40			;PRECEDENTES
2032	80	41	ADD	B	;ET ADDITION
2033	27	42	DAA		;CORRECTION DECIMALE
2034	32B120	43	STA	20B1H	;RANGEMENT SOMME UNITES
		44			;ET DIZAINES
2037	5F	45	MOV	E, A	;CHARGEMENT POUR AFFICH.
2038	3AB220	46	LDA	20B2H	;RAPPEL SOMME PRECEDENTE
		47			;CENTAINES ET MILLIERS
203B	CE00	48	ACI	00H	;ADDITION DE LA RETENUE
203D	27	49	DAA		;ET CORRECTION DECIMALE
203E	32B220	50	STA	20B2H	;RANGEMENT
2041	57	51	MOV	D, A	;CHARGEMENT POUR AFFICH.
2042	CD6303	52	CALL	UPDAD	;AFFICHAGE
2045	C30E20	53	JMP	BOUCLE	;ET REPRISE
		54			

— 202F : on appelle à ce moment le total cumulé précédent, stocké en 20B1. Il est nul au départ, puisqu'on a remis cette cellule à zéro. On ajoute le dernier opérande et, pour obtenir un résultat en décimal, on fait une correction DAA (en 2033). Le nouveau total est rangé, d'une part dans sa cellule mémoire attitrée, d'autre part dans le buffer qui est utilisé par le sous-programme d'affichage, le registre E.

— 2038 : l'addition précédente a pu donner naissance à une retenue pour les centaines. Dans ce cas, le bit indicateur de retenue, le «Carry» ou (CY) s'est positionné à 1. On va simuler une opération d'addition *avec retenue*, soit ACI, avec un second opérande nul (d'où ACI 00), le premier opérande étant le total cumulé pour les centaines et les milliers. Ceux-ci sont rangés à l'adresse 20B2 d'où on les appelle dans l'accumulateur. Au départ, 20B2 contient zéro. L'addition est à nouveau suivie d'une correction décimale, et du rangement de la nouvelle somme en 20B2 et dans le buffer d'affichage, le registre D. On exécute l'affichage (en 2042), et on peut recommencer.

Ce programme est simple à introduire et à exécuter. A propos, que se passerait-il si les quatre RLC qui commencent en 2012 étaient déplacés et introduits entre les instructions d'adresses actuelles 2021 et 2024 ?

SOMME DES n PREMIERS NOMBRES

But : Introduire la notion de boucle et de compteur.

Principales nouvelles instructions utilisées : SUB A, JNZ, DEC, INR

On se propose d'additionner les six premiers nombres, de 1 à 6 inclus, ce qui doit donner 21 en toute logique. Or, le microprocesseur qui travaille en binaire ne connaît pas le décimal : si on lui demande une telle addition, il fournira en résultat 15 ce qui, en hexadécimal, est parfaitement correct, mais moins évident pour l'opérateur.

Aussi, après chaque addition partielle va-t-on lui demander de rétablir le résultat en décimal, à l'aide de l'ordre spécifique DAA (*Decimal Adjust for Addition*), soit correction décimale).

La première instruction est un ordre de soustraction (SUB) du contenu de (A), sous-entendu...avec lui-même. On fait, en réalité : (A)-(A), ce qui ne peut donner que zéro : en effet, c'est là une autre façon élégante de remettre (A) à zéro car c'est dans l'accumulateur qu'on va stocker les résultats.

Puis, on charge le premier nombre, 1, dans le registre B. On va confier à (C) le soin de compter les additions et, puisqu'il doit y en avoir 6, on le charge immédiatement (MVI) avec 06.

La première addition (A) = 0 plus (B) = 1 donne 01 dans (A) ; l'ordre ADD B est suivi par cette fameuse correction décimale qui, ici, ne déclenchera aucune correction puisque tout va bien.

Après quoi, on passe au nombre suivant en incrémentant (B) de 1, soit INR B ; le registre B contient maintenant 2. On met ensuite à jour le compteur (C) en le décrémentant de 1 : DEC C, ce qui porte son contenu à 6 — 1 = 5. On voit que, lorsqu'il aura atteint zéro, cela signifiera que la liste des opérations est achevée ; si non, il faut poursuivre la tâche, ce que propose l'instruction JNZ BOUCLE : si *Non Zéro*, on se branche (J = *jump*, saut ou branchement) à l'instruction repérée par le mot-clé BOUCLE.

Tel est bien le cas à la première opération ; par conséquent, le programme va revenir à l'instruction «labellée» (ou «étiquetée») BOUCLE, d'adresse 2008 ; c'est bien cette adresse qu'on lit dans le code hexadécimal, sur la ligne JNZ BOUCLE : C2 08 20 ; à ceci près cependant, c'est qu'on a écrit C2 20 08 et non l'inverse.

En effet dans le cas du microprocesseur 8085 (et de sa famille), n'oublions pas qu'il faut intervertir les octets d'une adresse sur 16 bits dans le code opération. En effet, on commencera toujours par ranger l'octet de faible poids, puis celui de fort poids.

Au deuxième tour de boucle, on va ajouter (A) = 1 à (B) = 2 et obtenir (A) = 3. On incrémente encore B qui passe à 3 et on décrémente (C) qui passe à 4, et l'on procède à un 3ème tour.

L'addition de (A) = 3 avec (B) = 3 donne (A) = 6. En l'incrémentant, (B) passe à 4 et (C), décrémenté, à 3.

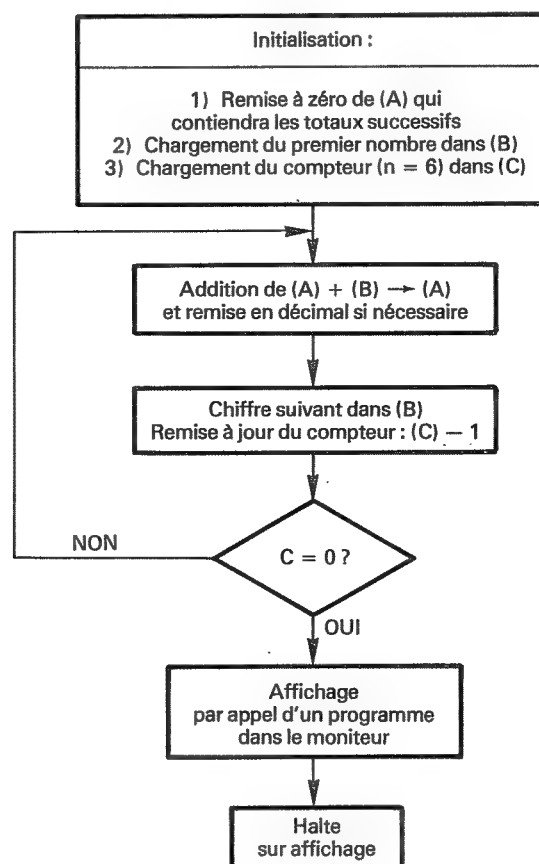
Au 4ème tour, (A) = 6 plus (B) = 4 donne...0A en hexadécimal. C'est à ce moment que l'ordre DAA va intervenir : voyant un résultat non souhaité, il va apporter une correction qui consiste à ajouter 6 à (A). Le contenu de (A) est alors de 0A + 06 = 10, ce qui, en décimal, est bien conforme à ce que l'on veut obtenir.

La 5ème boucle donne $(A) = 15$, sans correction (inutile ici), et la 6ème fournit, sans correction décimale $15 + 6 = 1B$, mais avec DAA (qui oblige à ajouter 06) : 21. Après cette correction, on a procédé à une dernière décrémentation de (C) qui arrive alors à zéro.

L'ordre de branchement si non zéro (JNZ), branchement conditionnel, n'est plus exécuté puisque la condition n'est plus respectée. Il est ignoré, et le microprocesseur va à l'instruction suivante : affichage du résultat contenu dans (A), en appelant le sous-programme spécifique du moniteur à l'adresse 036E labellé, UPDDT dans le moniteur.

L'affichage étant exécuté et afin qu'il soit maintenu (faute de quoi, on n'aurait pas le temps de le lire), on commande une «Halte» du microprocesseur qui stoppe toute opération.

L'introduction du programme se fait au clavier comme précédemment. Les paresseux remarqueront que, pour les codes numériques commençant par un zéro, il suffit de frapper le second digit et de faire NEXT ; en effet, la frappe du zéro de tête est inutile, le moniteur se chargeant de ce travail.



ISIS-II 8080/8085 MACRO ASSEMBLER, V4.0

SOMPRES PAGE 1

LOC	OBJ	LINE	SOURCE STATEMENT
		1	
		2	
		3	NAME SOMPRES
		4	
		5	;SOMME DES SIX PREMIERS NOMBRES
		6	
		7	

036E	8	UPDDT	EQU	036EH
	9			
	10			
2000	11	ORG	2000H	
	12			
2000 31C020	13	DEBUT: LXI	SP, 20C0H	
2003 97	14	SUB	A	; INITIALISATION
2004 0601	15	MVI	B, 01H	; NOMBRES SUCCESSIFS
2006 0E06	16	MVI	C, 06H	; COMPTEUR
2008 80	17	BOUCLE: ADD	B	
2009 27	18	DAA		; CORRECTION DECIMALE
200A 04	19	INR	B	; NOMBRE SUIVANT
200B 0D	20	DCR	C	; MISE A JOUR COMPTEUR
200C C20820	21	JNZ	BOUCLE	; SI (C)≠0 BOUCLER
200F CD6E03	22	CALL	UPDDT	; AFFICHAGE
2012 76	23	HLT		
	24			
2000	25	END	DEBUT	

LE PLUS GRAND DE DEUX NOMBRES

But : Exploiter les paires de registres, comprendre le rôle de la comparaison et de l'adressage indirect par registres.

Principales instructions utilisées : LXI, INX, DCX, CMP

On se propose de comparer deux nombres de deux digits hexadécimaux, logés en mémoire aux adresses 2040 et 2041, et de ranger le plus grand en 2042.

Toutes ces adresses sont assez astreignantes à traiter, aussi va-t-on loger celle de départ, 2040, dans deux registres d'un octet chacun mais mis bout à bout. En effet le 8085 permet le groupement de ses registres par paires, avec obligatoirement B et C, ou D et E, ou H et L. On va choisir H et L car cette paire dispose d'instructions spécifiques et intervient de façon très agréable dans l'adressage des données mémoire.

Le chargement *immédiat*, c'est-à-dire avec l'adresse qui suit aussitôt, s'écrit LXI ; le L vient de «load», chargement, alors que X précise qu'il s'agit d'une *paire de registres*, I venant pour *immédiat*. On indique de quelle paire de registres il s'agit : la paire H,L, en ajoutant simplement le nom du premier d'entre eux : l'ordre est donc LXI H, 2040. On remarquera à nouveau que, dans le code machine, 2040 est devenu 4020.

C'est au programme qu'on va demander de loger les deux nombres en mémoire, soit 4F pour l'un et 3B pour l'autre, mais l'adresse mémoire sera fournie désormais par (H, L).

Ainsi, l'ordre de mettre (MV, pour «Move», et I pour immédiat) la valeur 4F en M (c'est l'instruction MVI M, 4F), suffira : le microprocesseur décode cet ordre et sait que l'adresse de M est à prendre dans (H, L).

Cela fait, on va incrémenter de 1 le contenu de H, L qui passe alors à 2041. C'est l'ordre :

Adresses	Mémoire
2040	1er opérande : 4 F
2041	2ème opérande : 3 B
2042	le + grand des deux : donc 4 F

Occupation mémoire : les trois cellules de données

INX H, où X précise à nouveau qu'il s'agit d'une *paire de registres*, H indiquant que c'est la paire H, L qui est visée.

Un second ordre, MVI M, 3B stocke alors 3B en 2041. Puis, pour faire revenir (H, L) à 2040, on va décrémenter cette paire de 1, d'où l'ordre DCX H.

Dès lors, le programme proprement dit débute. On appelle le premier opérande, qui se trouve en 2040, et on le loge dans l'accumulateur A : c'est l'ordre MOV A,M. L'adresse M en mémoire est toujours indiquée par (H, L), et (M) est dupliqué, transféré dans (A).

On incrémente (H, L) qui passe à 2041 et on va comparer le contenu de (2041) à celui de (A) via l'ordre CMP M («comparer le contenu de la cellule mémoire pointée par H, L à, implicitement, le contenu de l'accumulateur»).

Cette opération de comparaison est un peu particulière. Elle consiste à faire (A) — (M), mais sans retenir le résultat : celui-ci va simplement provoquer le positionnement d'«indicateurs d'états» qui se trouvent dans le microprocesseur et qui diront si le résultat est *nul* (c'est l'*indicateur de résultat nul*, appelé *indicateur de zéro* qui intervient) ou s'il est *négatif* dans ce cas, une retenue apparaît à gauche des 8 bits, ce que l'*indicateur de retenue*, ou «*carry*», en anglais, détecte : voir la note «*Attention*» qui suit).

Ces indicateurs «surveillent» les opérations arithmétiques et logiques ; chacun d'entre eux a sa fonction propre, son état est le reflet du résultat de l'opération.

Par conséquent et s'il n'y a pas de retenue («*No Carry*»), c'est que (A) est supérieur ou égal (M) ; il restera à expédier (A) à l'adresse de rangement du plus grand nombre, soit 2042. C'est l'ordre de branchement conditionnel JNC, «*jump on No Carry*» qui provoque cette opération en renvoyant à l'instruction «*FAIT*» !

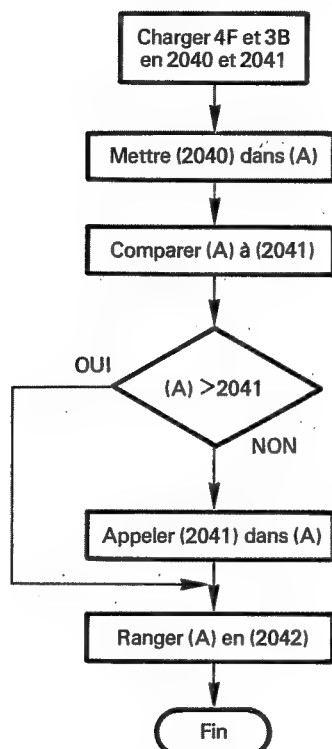
S'il y a eu une retenue, c'est que le second opérande est plus grand ; en conséquence, on l'appelle dans (A), on incrémente de toute façon (H, L) à l'instruction étiquetée FAIT pour pointer 2042, et on l'envoie à cette adresse.

Il suffira de faire RESET et d'aller lire le résultat en 2042 pour y trouver 4F.

A noter. — On a vu que les instructions MVI M, MOV A,M et MOV M,A se référaient à une adresse mémoire contenue dans la paire (H, L) plutôt que de donner immédiatement en clair cette adresse. Ainsi, avant d'aller en mémoire, il faut aller lire (H, L), ce que le microprocesseur fait d'ailleurs automatiquement dès qu'il a décodé l'instruction. C'est pourquoi l'on appelle ce processus un «*adressage indirect*» en précisant : *adressage indirect par registres*.

Attention ! — Autre remarque : l'usage de l'indicateur de retenue peut dérouter le lecteur qui se demande pourquoi on n'a pas, tout simplement, utilisé l'indicateur de *résultat négatif* puisque celui-ci existe. C'est tout simplement parce que ce dernier fonctionne en arithmétique à complément à 2 ; en effet, cet indicateur «*de signe*» («*Sign flag*») reprend le bit de fort poids d'un octet, interprété comme le signe avec un 1 pour le moins et un zéro pour le plus, et ne conviendrait donc absolument pas pour cette application.

Par contre, l'indicateur de retenue, lui, fonctionne ici à la perfection : posez une soustraction quelconque avec le diminueur plus grand que le diminuende ; vous constaterez qu'une retenue naît bel et bien à la gauche du bit de plus fort poids des octets.



Les adresses 2040, 2041, 2042 sont pointées dans la paire H, L

LOC	OBJ	LINE	SOURCE STATEMENT
		1	NAME TRI
		2	
2000		3	ORG 2000H
		4	
2000 214020		5	DEBUT: LXI H, 2040H ; RANGEMENT DES
2003 364F		6	MVI M, 4FH ; DEUX NOMBRES
2005 23		7	INX H ; EN
2006 363B		8	MVI M, 3BH ; MEMOIRE
2008 2B		9	DCX H ; APPEL DU PREMIER
2009 7E		10	MOV A, M
200A 23		11	INX H ; ET
200B BE		12	CMP M ; COMPARAISON
200C D21020		13	JNC FAIT ; SI (A) > (M) LE RANGER
200F 7E		14	MOV A, M ; SINON APPELER LE SECOND
2010 23		15	FAIT: INX H ; ET LE RANGER
2011 77		16	MOV M, A ; EN 2042H
2012 CF		17	RST 1
		18	
2000		19	END DEBUT

PUBLIC SYMBOLS

EXTERNAL SYMBOLS

USER SYMBOLS

DEBUT A 2000 FAIT A 2010

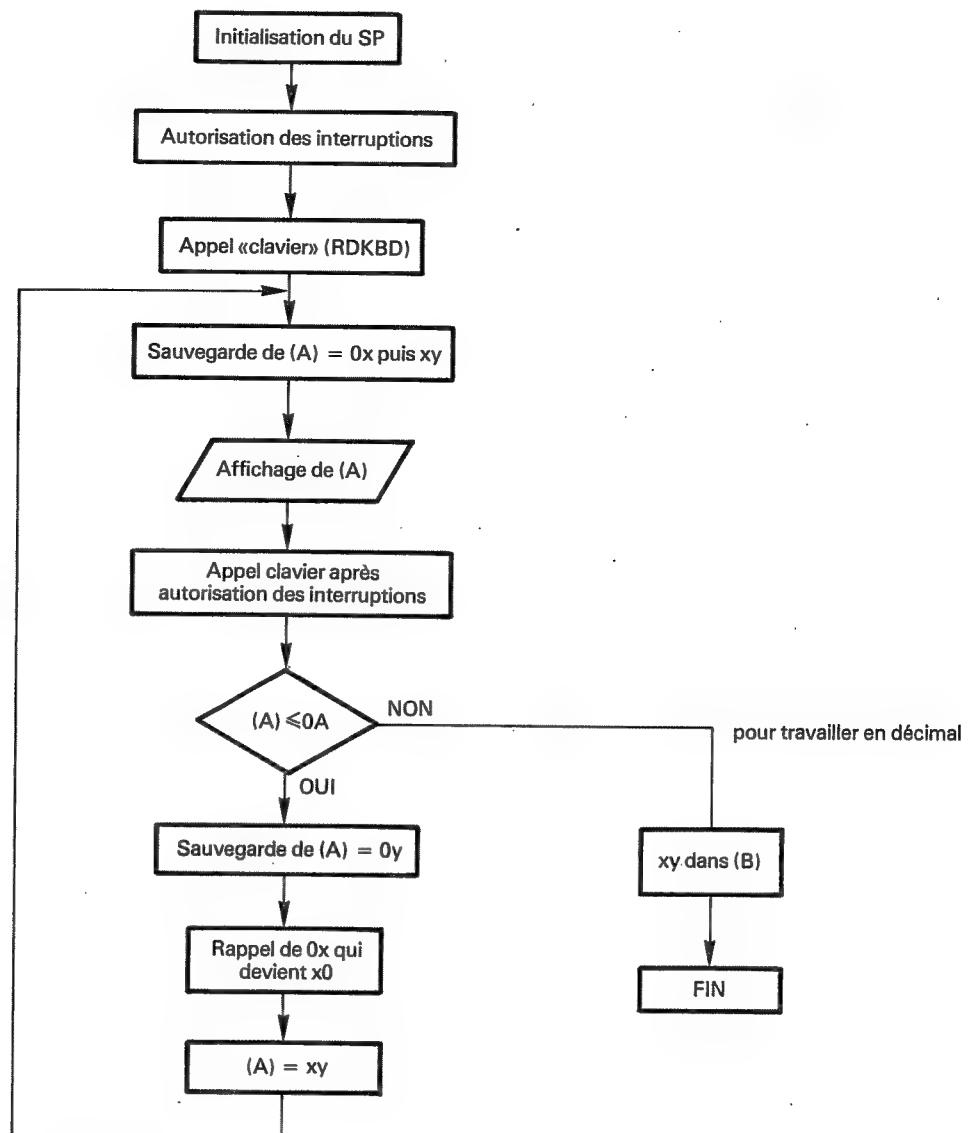
ASSEMBLY COMPLETE, NO ERRORS

ENTREE D'UN NOMBRE DE DEUX CHIFFRES AVEC AFFICHAGE

But : Utilisation des notions acquises au cours des programmes précédents.

Principales ou nouvelles instructions utilisées : RLC, STA, LDA

Problème - On veut entrer un nombre de deux chiffres dans A en l'affichant et en n'autorisant le stockage que si une touche représentant un chiffre est pressée. On sait que le programme RDKBD range dans A le code de la touche et que le programme UPDDT affiche le contenu de A, mais le détruit ainsi que les contenus des registres B, C, D, E, H et L ; il faut donc prévoir une sauvegarde d'un ou plusieurs registres



Le programme est donc le suivant :

```

1          .TITLE ENTREE
2
3          ;ENTREE D'UN NOMBRE DE DEUX CHIFFRES
4          ;    AVEC AFFICHAGE
5
6          .MACRO  SIM
7          .BYTE  030
8          .ENDM
9
10         02E7   RDKBD   =002E7
11         036E   UPDDT   =0036E
12
13 0000          . =02000
14
15 2000 31C020    LXI      SP,020C0
16 2003 3E08      MVI      A,008          ;AUTORISATION
17 2005          SIM          ;DES
18 2006 FB       EI          ;INTERRUPTIONS
19 2007 CDE702    CALL     RDKBD
20 200A F5        $1:      PUSH     PSW          ;SAUVEGARDE DE (A)
21 200B CD6E03    CALL     UPDDT          ;DETRUIT PAR "UPDDT"
22 200E FB       EI
23 200F CDE702    CALL     RDKBD
24 2012 00       NOP          ;OU RST 1 POUR VOIR ...
25 2013 FE0A      CPI      00A
26 2015 D22420    JNC      $0          ;SI (A)<0A CY=1
27 2018 47       MOV      B,A          ;SAUVEGARDE DE (A)
28 2019 F1       POP      PSW          ;RAPPEL NBRE PRECEDENT
29 201A E60F      ANI      00F          ;ON GARDE LE CHIFFRE
30 201C 07       RLC          ;DES UNITES
31 201D 07       RLC          ;QUI DEVIENT
32 201E 07       RLC          ;LE CHIFFRE
33 201F 07       RLC          ;DES DIZAINES
34 2020 80       ADD      B          ;FORME NOUVEAU NOMBRE
35 2021 C30A20    JMP      $1
36 2024 F1        $0:      POP      PSW
37 2025 47       MOV      B,A
38 2026 CF       RST      1          ;OU HLT,
39
40          ;LE MICROPROCESSEUR AFFICHE -8085 OU
41          ;NE RECOIT PLUS RIEN
42
43 0000          .END

```

Si PUSH et POP n'existaient pas, on pourrait effectuer la sauvegarde en utilisant une case mémoire «tampon», par exemple la case 2030 et on écrirait : STA 20 30, soit 32 30 20, à la place de PUSH et LDA 20 30, soit 3A 30 20, à la place de POP.

MULTIPLICATION DE DEUX NOMBRES POSITIFS DE DEUX CHIFFRES, ENTRES AU CLAVIER. LE RESULTAT EST AFFICHE

But : Synthèse des programmes précédents.

Dans ce programme, l'entrée des nombres est un sous-programme, ainsi que la conversion hexadécimal BCD et le calcul des centaines.

```

1          .TITLE MULDEC
2
3          ;MULTIPLICATION DECIMALE DE DEUX
4          ;NOMBRES POSITIFS DE DEUX CHIFFRES
5
6          .MACRO SIM
7          .BYTE 030
8          .ENDM
9
10         02E7   RDKBD   =002E7
11         0363   UPDAD   =00363
12         036E   UPDDT   =0036E
13
14 0000          . =02000
15
16 2000 31C020   LXI     SP,020C0
17 2003 3E08     MVI     A,008
18 2005         SIM
19 2006 CD3620 DEBUT: CALL   ENTREE
20 2009 47       MOV     B,A
21 200A C5       PUSH    B          ;"UPDDT" DETRUIT (B)
22 200B CD3620   CALL   ENTREE
23 200E C1       POP     B
24 200F 4F       MOV     C,A
25 2010 110000   LXI     DI,00000   ;"UPDDT" DETRUIT (DE)
26 2013 B8       CMP     B
27 2014 D21920   JNC     $0
28 2017 48       MOV     C,B
29 2018 47       MOV     B,A
30 2019 78       MOV     A,B
31 201A B7       ORA     A
32 201B CA2E20   JZ      FIN
33 201E E6F0     ANI     OF0
34 2020 C45B20   CNZ     CONVER
35 2023 79       MOV     A,C
36 2024 83       ADD     E
37 2025 27       DAA
38 2026 5F       MOV     E,A
39 2027 DC5520   CC      CENT
40 202A 05       DCR     B

```

```

41 202B C22320      JNZ      $1
42 202E CD6303 FIN:  CALL     UPDAD      ;SI (B)=0 PAS DE POINT
43                                     ;A DROITE DES CHIFFRES
44 2031 FB          EI
45 2032 76          HLT
46 2033 C30620      JMP      DEBUT
47
48                 ;ENTREE FORME LES NOMBRES DE DEUX CHIFFRES
49
50 2036 FB          ENTREE: EI
51 2037 CDE702      CALL     RDKBD
52 203A F5          $3:      PUSH     PSW
53 203B CD6E03      CALL     UPDDT
54 203E FB          EI
55 203F CDE702      CALL     RDKBD
56 2042 FE0A        CPI      00A
57 2044 D25320      JNC      $2
58 2047 4F          MOV      C,A
59 2048 F1          POP      PSW
60 2049 E60F        ANI      00F
61 204B 07          RLC
62 204C 07          RLC
63 204D 07          RLC
64 204E 07          RLC
65 204F 81          ADD      C
66 2050 C33A20      JMP      $3
67 2053 F1          $2:      POP      PSW
68 2054 C9          RET
69
70                 ;ON UTILISE C COMME CASE TAMPON CAR ON CHARGE B AU
71                 ;PREMIER APPEL ET CE DERNIER SERAIT DETRUIT AU
72                 ;DEUXIEME APPEL
73
74
75                 ;CENT INCREMENTE LES CENTAINES
76
77 2055 B7          CENT:    ORA      A
78 2056 14          INR      D
79 2057 7A          MOV      A,D
80 2058 27          DAA
81 2059 57          MOV      D,A
82 205A C9          RET
83
84                 ;CONVER CONVERTIT LE NOMBRE DANS B EN HEX
85
86 205B 07          CONVER:  RLC
87 205C 07          RLC
88 205D 07          RLC
89 205E 07          RLC
90 205F 5F          MOV      E,A
91 2060 78          MOV      A,B
92 2061 C6FA        $4:      ADI      0FA
93 2063 1D          DCR      E
94 2064 C26120      JNZ      $4
95 2067 47          MOV      B,A
96 2068 C9          RET
97
98      0000          .END

```

Le résultat apparaît en «adresse» avec le dernier nombre. On peut faire une nouvelle multiplication : il suffit de taper le nombre, ceci grâce à l'instruction HLT. Cette instruction nécessite l'autorisation des interruptions car elle met le microprocesseur en attente. *Ce dernier ne repart qu'après avoir traité une interruption* (s'il n'y a pas d'autorisation d'interruption, on ne repart que par un RESET).

Dans ce programme, une fois le résultat affiché, le microprocesseur est arrêté. On presse une touche, le 8279 fait une demande d'interruption (RST 5.5) ; le programme de gestion met le code de la touche dans la case mémoire 20FE et renvoie au programme principal où l'on trouve JMP «début», c'est-à-dire «Call RDKBD» ; ce programme va lire le contenu de la case 20FE (voir le programme «Entrée de données au clavier»).

MULTIPLICATION EN HEXADECIMAL DE NOMBRES ENTIERS NEGATIFS OU POSITIFS

*But: Apprendre à calculer avec des nombres réels en binaire.
Principales ou nouvelles instructions utilisées : JP adresse, ADC*

Nous avons travaillé jusqu'à maintenant avec des nombres positifs, mais nous aurons probablement à traiter des nombres négatifs ce qui complique un peu le problème, d'autant que certains microprocesseurs ne connaissent que les nombres signés c'est-à-dire où le bit de plus fort poids est réservé au signe (SC/MP par exemple).

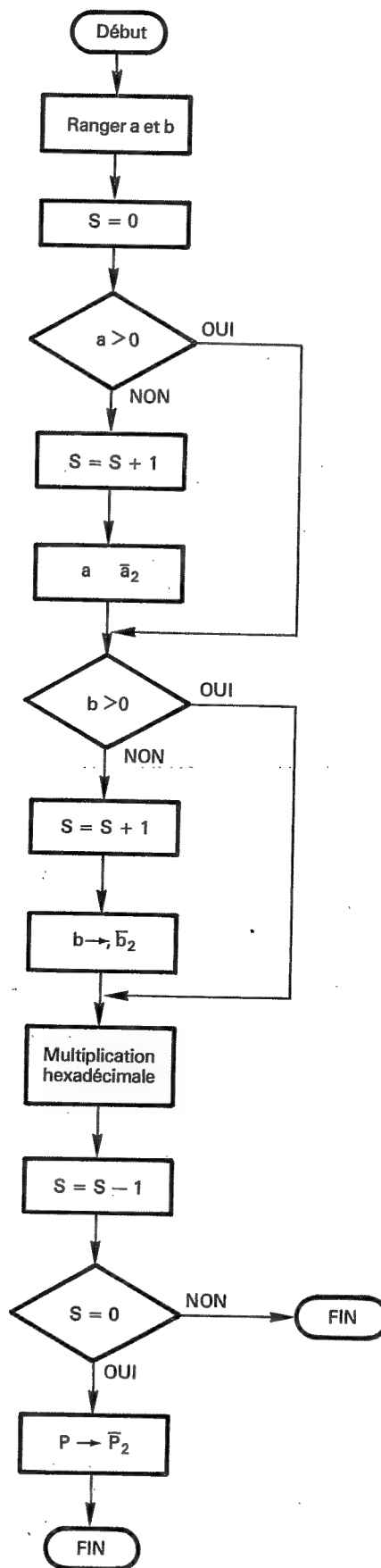
Si nous décidons que nos nombres seront signés (positifs ou négatifs), nous utilisons le bit de poids le plus fort comme *bit de signe*. Si ce bit est 0, le nombre est *positif* ; si ce bit est 1, le nombre est *négatif* et écrit sous la forme de son complément à 2 (ce qu'on a vu dans un programme précédent). Ainsi, si nous nous imposons des nombres de 8 bits maximum, nous serons limités à 7F, soit $+127_{10}$ pour les nombres positifs, et à 80, soit -128_{10} pour les nombres négatifs. En effet, FF est l'écriture de -1 :

FF	1111	1111
complément à 1	0000	0000
complément à 2	0000	0001
donc :	FF = -1_D	

et 80 est l'écriture de -128

80	1000	0000
complément à 1	0111	1111
complément à 2	1000	0000
donc :	80 = -128_D	

Dans notre multiplication, une fois les nombres rangés dans B et C, il faudra tester s'ils sont positifs ou négatifs. Dans le dernier cas, on mémorisera le signe et on convertira le nombre en positif. Le résultat sera traduit en nombre négatif s'il n'y a qu'un signe moins, ce qui donne l'organigramme suivant (où \bar{a}_2 est le complément à 2 de a) :



S est la mémorisation du signe, on voit que $S = 1$ signifie un résultat négatif : si $S = 0$ ou 2 , le résultat est positif. Nous n'avons pas détaillé la multiplication hexadécimale en raison de sa simplicité. Le programme est donc le suivant (les nombres sont entrés par programme, le résultat affiché) :

```

1          .TITLE MULHEX,'SIGNED'
2
3          ;MULTIPLICATION HEXADECIMALE SIGNED
4
5
6          0363    UPDAD    =00363
7
8          0000          . =02000
9
10
11 2000 31C020          LXI      SP,020C0          ;POUR AFFICHAGE
12 2003 01CDAB          LXI      B,0ABCD          ;ENTREE DES NOMBRES
13 2006 110000          LXI      D,00000          ;INITIALISATION
14 2009 6B              MOV      L,E              ;      "      S=0
15 200A 78              MOV      A,B
16 200B B7              ORA      A              ;"FLAGS" POSITIONNES
17 200C F21220          JP      $0              ;TEST DU BIT DE SIGNE
18 200F 2C              INR      L              ;S=S+1
19 2010 2F              CMA              ;COMPLEMENT
20 2011 3C              INR      A              ;A 2 DE "AB"
21 2012 47              MOV      B,A
22 2013 79              MOV      A,C
23 2014 B7              ORA      A
24 2015 F21B20          JP      $1
25 2018 2C              INR      L
26 2019 2F              CMA              ;COMPLEMENT
27 201A 3C              INR      A              ;A 2 DE "CD"
28 201B 4F              MOV      C,A
29 201C B8              CMP      B              ;"CD" EST DANS A
30                                ;SI "CD" < "AB" : CY=1
31 201D D22220          JNC      $2
32 2020 48              MOV      C,B              ;(C) = LE PLUS GRAND
33 2021 47              MOV      B,A              ;(B) = LE PLUS PETIT
34 2022 7B              MOV      A,E
35 2023 81              BOUCLE: ADD      C
36 2024 D22B20          JNC      $3
37 2027 14              INR      D
38 2028 05              DCR      B
39 2029 C22320          JNZ      BOUCLE
40 202C 5F              MOV      E,A
41 202D 2D              DCR      L              ;SI (L)=0 RESULTAT < 0
42 202E C23B20          JNZ      FIN
43 2031 7B              MOV      A,E              ;COMPLEMENT
44 2032 2F              CMA              ;A 2 DE (E)
45 2033 C401          ADI      001              ;INR A NE MARCHE PAS
46 2035 5F              MOV      E,A              ;CAR CY NON AFFECTE
47 2036 7A              MOV      A,D              ;COMPLEMENT
48 2037 2F              CMA              ;A 1 DE (D)
49 2038 CE00          ACI      000              ;POUR LE CAS OU CY = 1
50 203A 57              MOV      D,A              ;APRES COMP. DE (E)
51 203B CD6303 FIN:    CALL      UPDAD
52 203E 76              HLT

```

Exemple 1 :

$a = 52_D =$	0011 0100 _B =	34 _H
$b = 107_D =$	0110 1011 _B =	6B _H
$P = 5564_D =$	0001 0101 1011 1100 _B =	15BC _H

Exemple 2 :

$a = -52_D =$	1100 1100 _B =	CC _H
$b = 107_D =$	0110 1011 _B =	6B _H
$P = -5564_D =$	1110 1010 0100 0100 _B =	EA44 _H

On peut remarquer que si l'on fait la multiplication non signée de CC_H par 6B_H, on n'obtient pas le bon résultat car on multiplie 204_D par 107_D, on obtient ici un nombre *non signé* où le 16ème bit est 0 (CC × 6B = 5544_H)

Exemple 3 :

$a = -52_D =$	1100 1100 _B =	CC _H
$b = -107_D =$	1001 0101 _B =	95 _H
$P = +5564_D =$	0001 0101 1011 1100 _B =	15BC _H

(avec même remarque que précédemment)

Exemple 4 :

$a = +44_D =$	0010 1100 _B =	2C _H
$b = -128_D =$	1000 0000 _B =	80 _H
$P = -5632_D =$	1110 1010 0000 0000 _B =	EA00 _H

Ici, le résultat avant conversion en complément à 2 est 0001 0110 0000 0000 (soit 1600_H) ; si l'on n'avait pas pensé à tenir compte de la retenue («carry») lors de la complémentation du contenu de D, on aurait E900, c'est-à-dire — 5888. En effet, 5632 = 0001 0110 0000 0000 avec l'octet de fort poids dans D et celui de faible poids dans E ; si on complémente le contenu de E à 2, on obtient (1) 0000 0000 avec (1) dans la retenue.

ADDITION SUR 16 BITS

But : Développer l'usage des paires de registres et de quelques ordres qui leur sont spécifiques.

Principales ou nouvelles instructions utilisées : XCHG, DAD, LHLD, SHLD.

Un certain nombre d'instructions spécifiques portent sur des *paires de registres* comme on l'a déjà vu. Pour en maîtriser mieux l'usage, on va exécuter une addition sur 16 bits à l'aide de ces paires.

Le cumulande doit au préalable être rangé en mémoire, en 2040 et 2041 ; supposant qu'on adopte 3AB2 pour valeur. On va lui additionner, par exemple, B104, logé en 2042 et 2043. Ce rangement étant fait :

- On appelle le cumulande dans la paire D, E ;
- Puis on le transfère dans H, L à l'aide de l'ordre d'échange croisé XCHG : (D, E) passe dans (H, L) et à l'inverse, (H, L) va dans (D, E), ce deuxième mouvement n'offrant ici aucune importance pour nous.
- On envoie ensuite le cumulateur dans (D, E).

Il ne reste plus qu'à additionner (D, E) à (H, L), ce qui se fait à l'aide de l'ordre DAD D. Le résultat, qui reste dans (H, L), sera, si on le veut, expédié en 2044 et 2045.

On remarquera que les ordres de chargement et de rangement du contenu de la paire H, L (LHLD : *load HL direct*, soit chargement de HL en adressage direct ; et SHLD ; *store HL direct*) se bornent à fournir l'adresse de la première cellule mémoire visée, le microprocesseur interprétant LHLD et SHLD en allant visiter réellement deux cellules consécutives.

L'occupation mémoire

Cumulande = 3AB2 dans 2040 et 2041

2040 = 3A
2041 = B2

Cumulateur = B104 dans 2042 et 2043

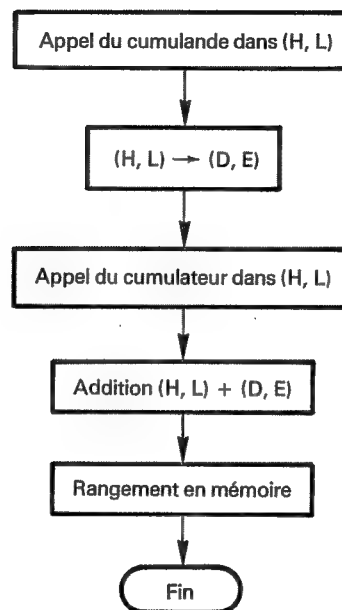
2042 = B1
2043 = 04

Somme = EBB6 dans 2044 et 2045

2044 = EB
2045 = B6

Ce programme se termine par une «Halte» ; et pourquoi pas, puisqu'à ce moment, on stoppe le microprocesseur qui doit attendre la suite. Dans ce cas, l'affichage ne revient pas à «— 8085» mais reste sur un «E», à sa gauche, pour marquer qu'il est en «exécution» (même s'il attend la suite). Il faudra donc faire un RESET, puis lire en 2044 et 2045 le résultat de l'opération qui devrait, en toute logique, donner EBB6.

Avant de lancer ce programme, n'oubliez pas de ranger au préalable les deux nombres de 16 bits aux adresses 2040, 2041 et 2042, 2043.



LOC	OBJ	LINE	SOURCE STATEMENT
		1	NAME ADDHEXA16
		2	
		3	; ADDITION HEXADECIMALE SUR 16 BITS
		4	
		5	
		6	; CHARGER DEUX NOMBRES DE 16 BITS EN 2040, 2042
		7	
2000		8	ORG 2000H
		9	
2000	2A4020	10	DEBUT: LHLD 2040H ; 1ER OPERANDE DANS (H, L)
2003	EB	11	XCHG ; TRANSFERT DANS (D, E)
2004	2A4220	12	LHLD 2042H ; 2EME OPERANDE DANS (H, L)
2007	19	13	DAD D ; ET ADDITION
2008	224420	14	SHLD 2044H ; RANGEMENT
200B	76	15	HLT
		16	
2000		17	END DEBUT

PUBLIC SYMBOLS

EXTERNAL SYMBOLS

USER SYMBOLS

DEBUT A 2000

ASSEMBLY COMPLETE, NO ERRORS

CONVERSION HEXADECIMAL - BCD POUR NOMBRES ENTIERS

But : Convertir un nombre hexadécimal en décimal codé binaire.

Principales ou nouvelles instructions : DCX, RRC.

On a souvent intérêt à effectuer la totalité d'un calcul et à n'exprimer le résultat en décimal codé binaire, BDC ou BCD) qu'à la fin. Ce programme convertit un nombre de 4 digits, inférieur ou égal à 270F, en hexadécimal, en un nombre de 4 chiffres, inférieur ou égal à 9999, en décimal. Le principe adopté est le suivant.

1)- On convertit le chiffre des unités en lui ajoutant 6 s'il dépasse 9.

2)- On ajoute au nombre autant de fois 16 qu'il comporte de «seizaines».

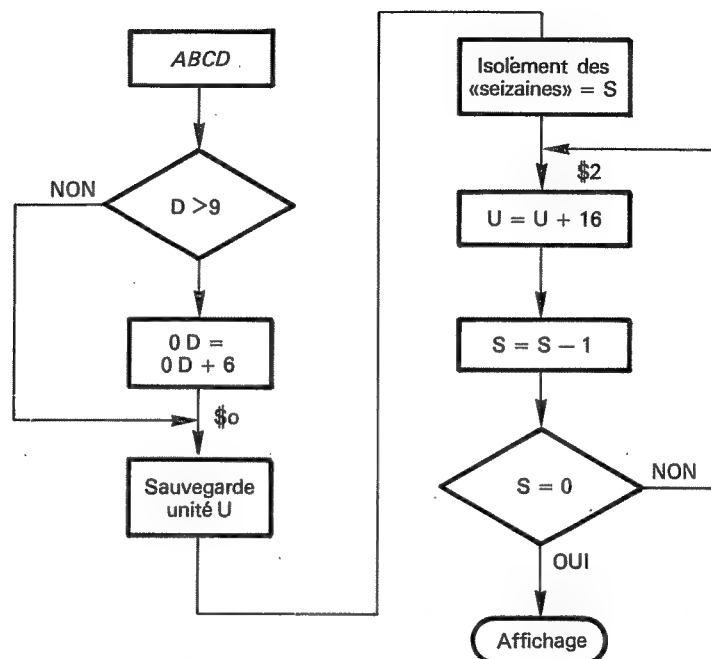
Cela donne l'organigramme suivant, en supposant que le nombre hexadécimal est «ABCD», qu'il est contenu dans la paire HL et que le résultat est rangé dans la paire DE pour affichage («Call UPDAD»).

```

1      .TITLE CONVER,'HEXA-DCB'
2
3      ;CONVERSION HEXA-DCB POUR NOMBRES ENTIERS
4
5
6      0363    UPDAD    =00363
7
8
9      0000      . =02000
10
11     2000 31C020      LXI      SP,020C0
12     2003 1600      MVI      D,000      ;(D)=0 POUR CAS OU
13                                     ;CENTAINES = 0
14     2005 21CDAB      LXI      H,0ABCD
15     2008 7D      MOV      A,L
16     2009 E60F      ANI      00F      ;ON ISOLE "D"
17     200B FE0A      CPI      00A      ;SI "D"<=9 ALORS CY=1
18     200D DA1120      JC      $0
19     2010 27      DAA
20     2011 5F      $0:      MOV      E,A      ;ON AJOUTE 6 SI "D">9
21                                     ;SAUVEGARDE DES UNITES
22     2012 7D      MOV      A,L      ;ET DES DIZAINES
23     2013 E6F0      ANI      0F0      ;ISOLEMENT DE "C"
24     2015 0F      RRC
25     2016 0F      RRC
26     2017 0F      RRC
27     2018 0F      RRC
28     2019 4F      MOV      C,A      ;C CONTIENT "0D"
29     201A 7C      MOV      A,H
30     201B E60F      ANI      00F      ;ISOLEMENT DE "B"
31     201D 07      RLC      ;ON PEUT
32     201E 07      RLC      ;FAIRE
33     201F 07      RLC      ;EGALEMENT
34     2020 07      RLC      ;4 RRC
35     2021 81      ADD      C
36     2022 4F      MOV      C,A      ;A CONTIENT "BC"
                                     ;C CONTIENT "BC"

```

37	2023	7C		MOV	A,H	
38	2024	E6F0		ANI	0F0	ISOLEMENT DE "A"
39	2026	07		RLC		
40	2027	07		RLC		
41	2028	07		RLC		
42	2029	07		RLC		
43	202A	47		MOV	B,A	B CONTIENT "0A" DONC
44						(BC) = "0ABC" C'EST A
45						DIRE LES "SEIZAINES"
46	202B	B1		ORA	C	"0A" = "BC" = 0 ?
47	202C	CA4220		JZ	FIN	
48	202F	7B	\$1:	MOV	A,E	
49	2030	C616		ADI	016	
50	2032	27		DAA		
51	2033	5F		MOV	E,A	
52	2034	D23C20		JNC	\$2	SI CY=1 CALCUL DES
53						CENTAINES
54	2037	B7		ORA	A	CY=0
55	2038	14		INR	D	
56	2039	7A		MOV	A,D	
57	203A	27		DAA		
58	203B	57		MOV	D,A	
59	203C	0B	\$2:	DCX	B	"DCX" N'AFFECTE PAS
60						LES "FLAGS"
61	203D	79		MOV	A,C	
62	203E	B0		ORA	B	ON AURA 0 SI
63						(B) = (A) = (C) = 0
64	203F	C22F20		JNZ	\$1	
65	2042	CD6303	FIN:	CALL	UPDAD	PAS DE POINT CAR (B)=0
66	2045	76		HLT		
67						
68		0000		.END		



Remarque technique.- Lors de l'exécution de l'instruction DCX, le contenu du registre de poids le plus fort (B, D, H) se retrouve momentanément sur le bus des *adresses hautes*. et peut causer des sélections accidentelles de circuits... dont il faudra se méfier.

CONVERSION BCD - HEXADECIMAL POUR NOMBRES ENTIERS

But : Convertir un nombre de 4 chiffres décimaux codé BCD en hexadécimal.

Principales ou nouvelles instructions utilisées : DAD p, XCHG.

Ce programme traduisant un nombre BCD en hexadécimal sera appelé avant une opération arithmétique en binaire. Il est le dual du précédent.

Nous avons vu, dans un programme antérieur de «Conversion DCB - hexadécimal de deux nombres de un chiffre» que, pour convertir un nombre BCD de deux chiffres en un nombre hexadécimal, il faut lui retrancher autant de fois 6 qu'il comporte de dizaines.

Par exemple :

$$(AB - 6 \times A) = AB_H : AB \text{ en hexadécimal}$$

Par analogie, on voit qu'il faudra retrancher autant de fois 156 ($16^2 - 10^2$) qu'il y aura de centaines et autant de fois 3096 ($16^3 - 10^3$) qu'il y aura de milliers :

$$(ABCD - A \times 3096 - B \times 156 - C \times 6) = ABCD_H$$

```

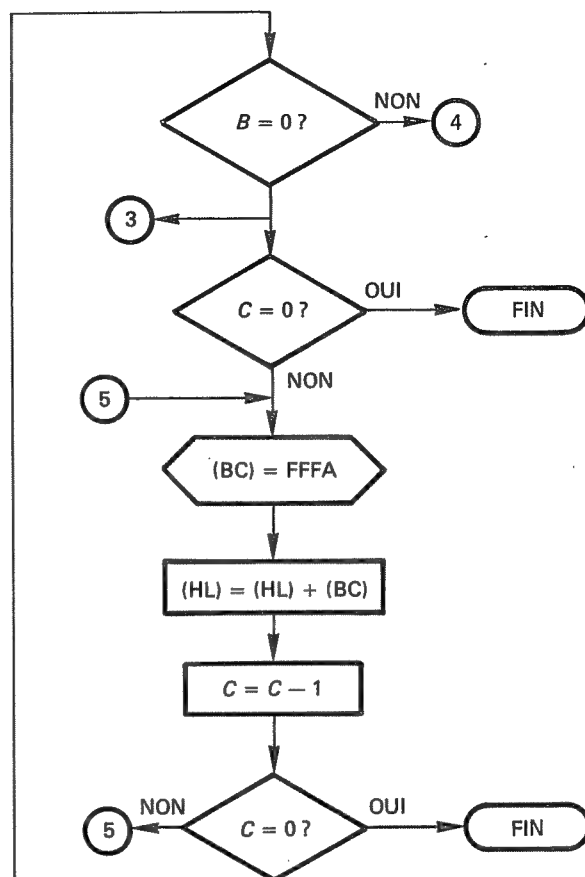
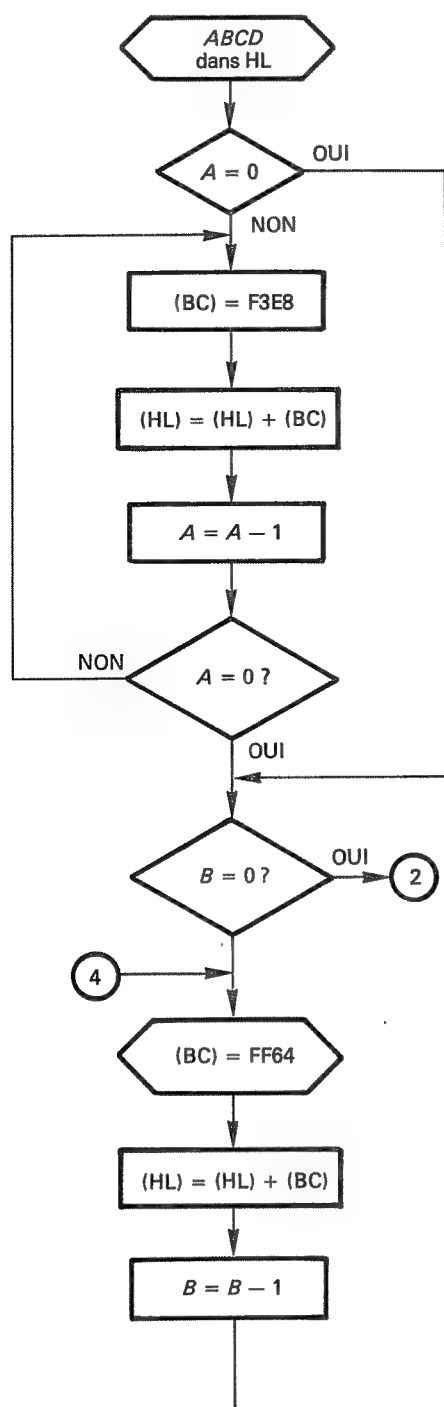
1          .TITLE CONVER,'DCB-HEX-1'
2
3          ;CONVERSION DCB-HEXA POUR NOMBRES ENTIERS
4          ;   VERSION 1
5
6          0363    UPDAD    =00363
7
8          0000          . =02000
9
10         2000 21CDAB"    LXI      H,0ABCD
11         2003 31C020    LXI      SP,020C0    ;POUR AFFICHAGE
12         2006 54        MOV      D,H        ;SAUVEGARDE DE "AB"
13         2007 5D        MOV      E,L        ;ET DE "CD"
14         2008 7A        MOV      A,D        ;ISOLEMENT
15         2009 E6F0      ANI      OF0        ;DE
16         200B CA1A20    JZ       $1        ;"A"
17         200E 0F        RRC
18         200F 0F        RRC
19         2010 0F        RRC
20         2011 0F        RRC
21         2012 01E8F3    LXI      B,0F3E8
22         2015 09        $2:      DAD      B        ;"ABCD"="ABCD"+F3E8
23         2016 3D        DCR      A
24         2017 C21520    JNZ      $2
25         201A 7A        $1:      MOV      A,D        ;ISOLEMENT DE "B"
26         201B E60F      ANI      00F

```

27	201D	CA2820	JZ	\$3	
28	2020	0164FF	LXI	B,OFF64	
29	2023	09	\$4:	DAD	B
30	2024	3D	DCR	A	
31	2025	C22320	JNZ	\$4	
32	2028	7B	\$3:	MOV	A,E
33	2029	E6F0	ANI	OF0	; ISOLEMENT DE "C"
34	202B	CA3A20	JZ	\$5	
35	202E	0F	RRC		
36	202F	0F	RRC		
37	2030	0F	RRC		
38	2031	0F	RRC		
39	2032	01FAFF	LXI	B,OFFFA	
40	2035	09	\$6:	DAD	B
41	2036	3D	DCR	A	
42	2037	C23520	JNZ	\$6	
43	203A	EB	\$5:	XCHG	; ECHANGE (DE) ET (HL)
44	203B	CD6303	CALL	UPDAD	
45	203E	76	HLT		
46					
47		0000	.END		

1			.TITLE CONVER,'DCB-HEXA-2'	
2				
3			;CONVERSION DCB-HEXA POUR NOMBRES ENTIERS	
4			; VERSION 2	
5				
6		0363	UPDAD	=00363
7				
8		0000		. =02000
9				
10	2000	21CDAB	LXI	H,0ABCD
11	2003	31C020	LXI	SP,020C0
12	2006	54	MOV	D,H
13	2007	5D	MOV	E,L
14	2008	7A	MOV	A,D
15	2009	E6F0	ANI	OF0
16	200B	01E8F3	LXI	B,OF3E8
17	200E	C42820	CNZ	CLLG
18				;SI "A"=0 ON APPELLE
19	2011	7A	MOV	A,D
20	2012	E60F	ANI	00F
21	2014	0164FF	LXI	B,OFF64
22	2017	C42C20	CNZ	CLCT
23				;SI "B"=0 ON APPELLE
24	201A	7B	MOV	A,E
25	201B	E6F0	ANI	OF0
26	201D	01FAFF	LXI	B,OFFFA
27	2020	C42820	CNZ	CLLG
28	2023	EB	XCHG	
29	2024	CD6303	CALL	UPDAD
30	2027	76	HLT	

31				
32	2028	0F	CLLG:	RRC
33	2029	0F		RRC
34	202A	0F		RRC
35	202B	0F		RRC
36	202C	09	CLCT:	IAD
37	202D	3D		DCR
38	202E	C22C20		JNZ
39	2031	C9		RET
40				
41		0000		.END



Ces opérations de soustraction doivent être exécutées sur des nombres de 4 chiffres, c'est-à-dire sur une paire de registres.

Dans le jeu d'instructions, il n'existe que trois opérations sur 16 bits : `INX p`, `DCX p`, `DAD p`. Cette dernière réalise l'addition sur 16 bits dans la paire HL, c'est-à-dire que `DAD B` nous donnera : $(HL) = (HL) + (BC)$, le contenu de la paire B étant conservé. Nous utiliserons donc cette instruction et pour ce faire écrirons 6, 156, 3096 sous la forme de leur complément à 2, soit respectivement FFFA, FF64, F3E8.

Le nombre est, à l'origine, dans HL ; le résultat sera dans HL. Pour l'affichage éventuel, on doit utiliser la paire DE. Pour transférer le contenu de HL dans DE, on utilise une instruction `XCHG (EB)` qui échange les contenus de H et D d'une part et de L et E d'autre part.

Le programme a été écrit (version 1) sans « astuce » ; or, en réfléchissant, on arrive à gagner 6 octets (rotation et boucle en sous-programme). Mais avec un sous-programme unique, ayant deux entrées possibles, on obtient la version 2 avec un gain de 13 octets !

CONVERSION BCD - HEXADECIMAL ET HEXADECIMAL-BCD POUR NOMBRES NON ENTIERS

But : Etendre les programmes précédents aux nombres à «virgule» - Test de AC dans la soustraction BCD.

Principales ou nouvelles instructions utilisées : CMP M, SUB, SUI, LDAX p, RAL.

La conversion des nombres entiers est relativement simple puisqu'un chiffre correspond à un nombre entier de fois la base.

Exemple (avec B = binaire, H = hexadécimal, D = BCD) :

$$70_D = 7 \times 10^1 = 46_H = 4 \times 16^1 + 6 \times 16^0 = 0100\ 0110_B$$

$$= 1 \times 2^6 + 1 \times 2^2 + 1 \times 2^1$$

Dans le cas des chiffres après la virgule cela se complique.

Exemple :

$$0,1_H \quad \frac{1}{16} = 0,0625_D$$

$$0,1_B \quad \frac{1}{2} = 0,50_D$$

$$0,11_H = \frac{1}{16} + \frac{1}{(16)2} = 0,0625 + 0,0039 = 0,0664_D$$

$$0,11_D = \frac{1}{2} + \frac{1}{22} = 0,75_D$$

Compte tenu de la complexité du problème nous nous limiterons à un nombre binaire «après-la-virgule» de 8 bits dont les poids respectifs sont donnés par le tableau :

Binaire	Fraction décimal	Décimal
,1	1/2	,5
,01	1/4	,25
,001	1/8	,125
,0001	1/16	,0625
,00001	1/32	,03125
,000001	1/64	,015625
,0000001	1/128	,0078125
,00000001	1/256	,00390625

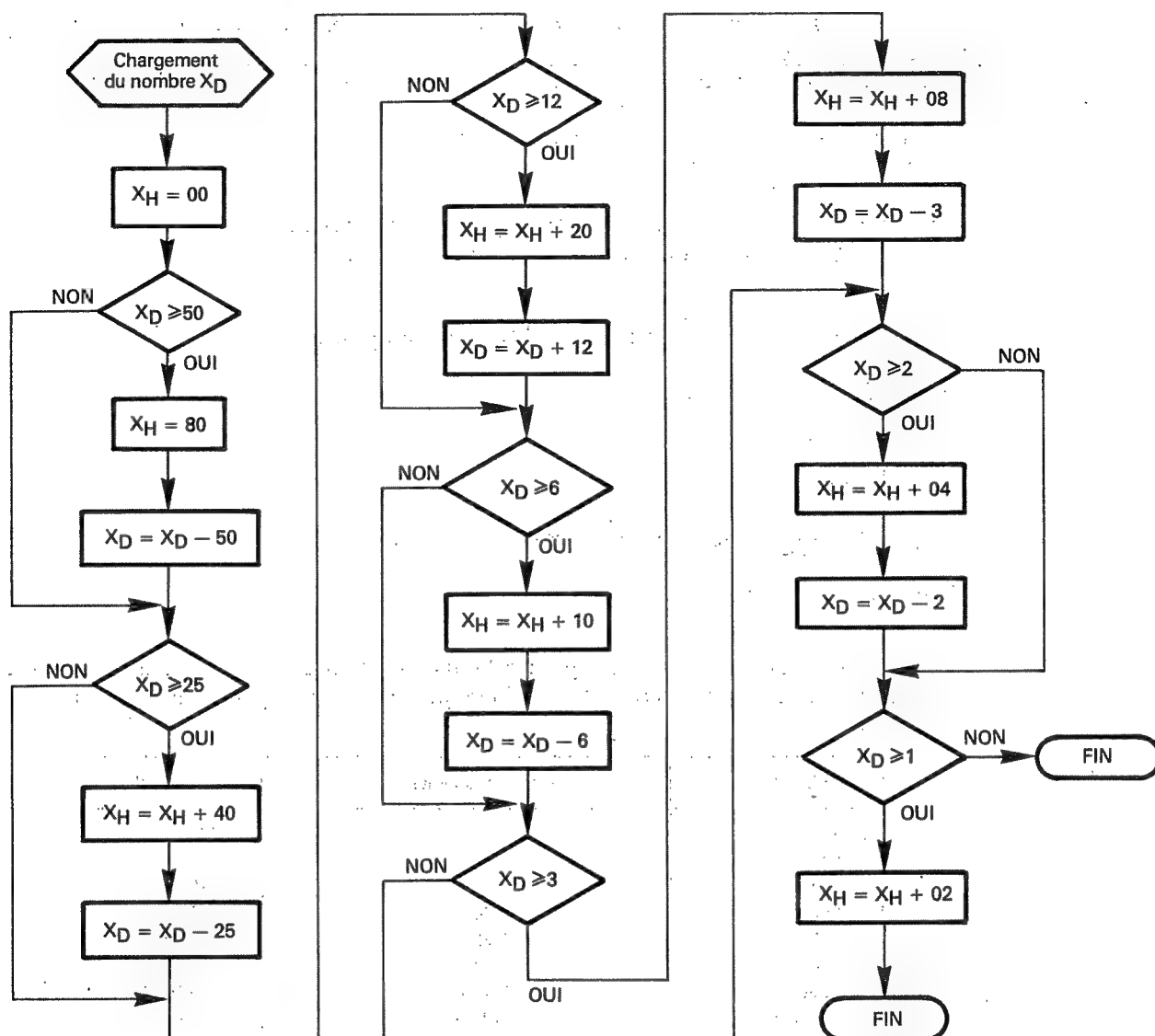
Nous voyons immédiatement que l'expression en BCD devrait prendre énormément de place, aussi nous limiterons-nous à deux chiffres après la virgule (en BCD) en utilisant le tableau ci-dessous :

Binaire	Décimal	Hexadécimal
0,10000000	0,50	0,80
0,01000000	0,25	0,40
0,00100000	0,12	0,20
0,00010000	0,06	0,10
0,00001000	0,03	0,08
0,00000100	0,02	0,04
0,00000010	0,01	0,02
0,00000001	0,00	0,01

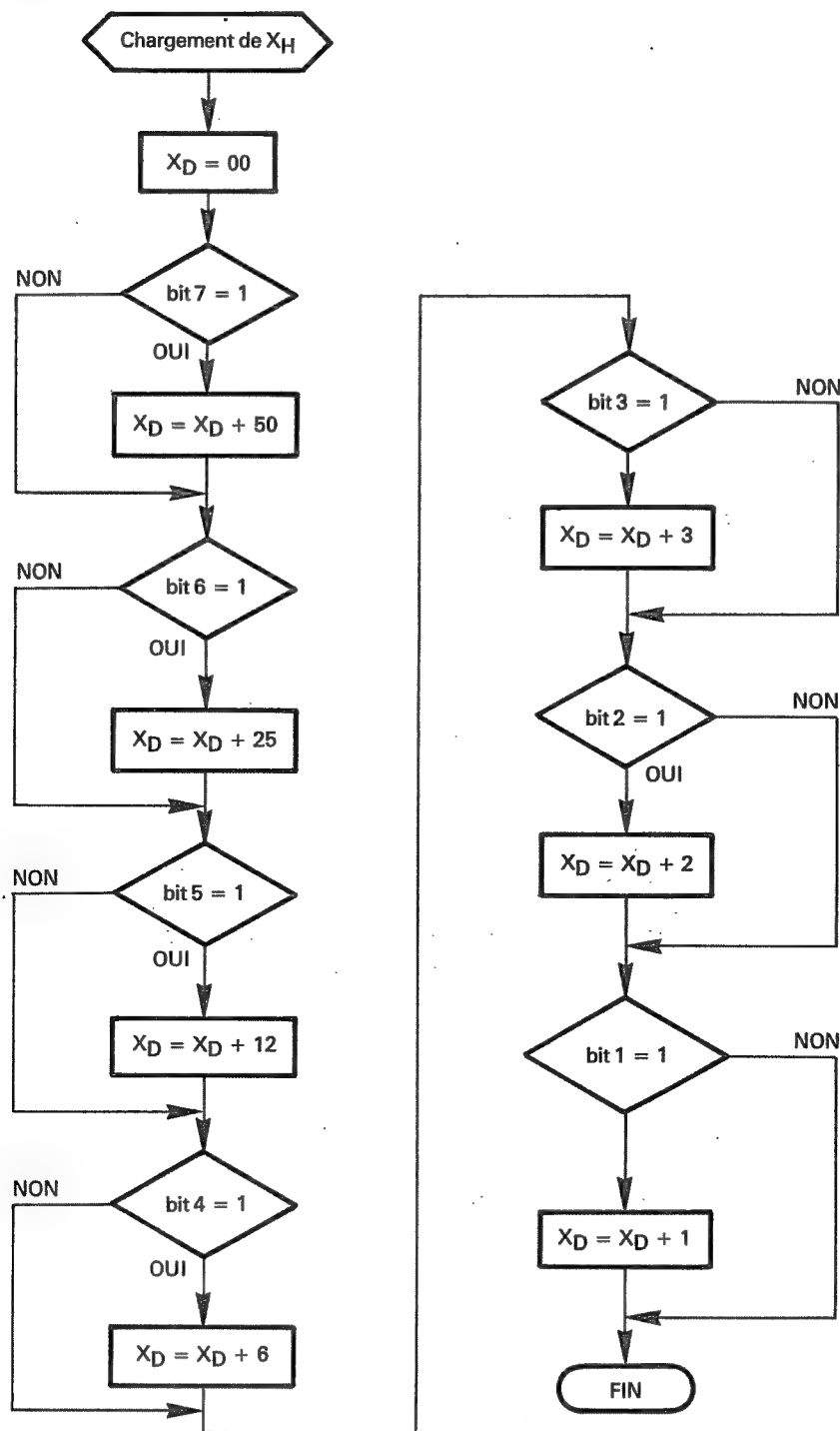
Ainsi : $0,FE = 0,99$

Pour convertir un nombre d'une base dans l'autre, il sera plus efficace d'utiliser la tableau ci-dessus plutôt que le calcul, ce qui nous conduit aux organigrammes suivants :

Conversion BCD - hexadécimal



Conversion hexadécimal — BCD



De tels organigrammes donnent l'impression que le programme sera long, mais en utilisant les instructions du 8085, on écrit des programmes compacts. Ainsi, on passe de 80 à 40 puis à 20, etc., par *rotation à droite sans retenue (carry)* donc, si on range 80 dans un registre et que l'on effectue une rotation à chaque test, le programme de conversion BCD-hexadécimal se résumera en une boucle dans laquelle il y aura des rotations pour passer successivement de 80 à 02. Les comparaisons par rapport à 50, 25 etc. seront effectuées par rapport à ces nombres préalablement rangés en mémoire.

Pour le programme de conversion hexadécimal-BCD, le test sur les bits se fera par *rotation à gauche avec Carry (RAL)* et on testera le «carry». Le nombre à ajouter à X_D sera pris dans la pile précédemment définie.

Une difficulté apparaît : il s'agit de la soustraction BCD. Nous devons soustraire 50, 25, etc., et retrouver un nombre écrit en BCD. Nous ne pouvons utiliser l'ordre DAA *et il faut donc parfaitement comprendre le fonctionnement du microprocesseur*. Prenons quelques exemples :

— Exemple. - $X_D = 83 = 1000\ 0011$ au BCD. Soustrayons 50 ; d'après ce qui a été vu antérieurement :

$$\begin{array}{rcl} 50 & = & 0101\ 0000 \quad \text{donc} \\ - 50 & = & 1011\ 0000 \quad \text{et} \\ \text{Donc:} & & \\ 83 & = & 1000\ 0011 \\ - 50 & = & 1011\ 0000 \\ \hline 33 & = & (1)\ 0011\ 0011 \end{array}$$

Ceci qui nous convient. Soustrayons maintenant 25 de ce résultat :

$$\begin{array}{rcl} 25 & = & 0010\ 0101 \\ - 25 & = & 1101\ 1011 \\ \text{Donc:} & & \\ 33 & = & 0011\ 0011 \\ - 25 & = & 1101\ 1011 \\ \hline 08 & = & (1)\ 0000\ 1110 \end{array}$$

Rien ne va plus, car si l'on tente un DAA, il y a une retenue ; mais nous sommes en *soustraction* : le microprocesseur, dans ce cas, *complémente le «carry»* appelé ici *«borrow»* («retenue de la soustraction») ; il n'ajoutera donc pas 60 (ouf !) mais le quartet bas est supérieur à 9 et il n'y a pas de «carry» auxiliaire : il ajoutera donc 06 :

$$\begin{array}{r} 0000\ 1110 \\ + 0000\ 0110 \\ \hline 0001\ 0100 \end{array}$$

Le résultat est toujours différent de 8. Comment passer du 0000 1110 à 0000 1000 ? Nous voyons qu'il faut *soustraire* 0000 0110 (06), c'est-à-dire ajouter 1111 1010 (soit FA) ; en effet :

$$\begin{array}{r} 0000\ 1110 \\ 1111\ 1010 \\ \hline (1)\ 0000\ 1000 = 08 \end{array}$$

Il faut maintenant voir dans quel cas nous ajouterons FA (soustrairons 06), donc comprendre le fonctionnement du 8085.

Nous avons écrit directement le complément à 2 du nombre à soustraire ; or, l'opération se fait en deux temps : complément à 1, *puis* on ajoute 1. Cette opération affecte les indicateurs (*flags*), en particulier le «carry» auxiliaire. Nous aurons donc :

$$\begin{array}{rcl} & \boxed{1}\ 1111 & \text{(retenues)} \\ 88 & = & 1000\ 0011 \\ - 50 & = & 1010\ 1111 \quad \text{(complément à 1)} \\ & + & 1 \quad \text{(complément à 2)} \\ \hline 33 & = & 10011\ 0011 \end{array}$$

La retenue encadrée est AC («carry» auxiliaire) ; donc $83-50 = 33$ avec $AC = 1$ et B («borrow») = 0 Puis :

		11	11	
33	=	0011	0011	retenues
-25	=	1101	1010	
	+		1	
		0000 1110		

Ici, il n'y a pas de AC : $AC = 0$, on devra donc retrancher 6.

— *Problème.* — Comment tester AC alors qu'il est normalement inaccessible ? Il faut amener le registre des indicateurs (flags) dans A et tester le 4ème bit. Une telle opération se fera à l'aide d'un PUSH et d'un POP (adresses 2012 et 2013).

```

1      .TITLE CONVER,'DCB-HEX DEC'
2
3      ;CONVERSION DCB-HEXA POUR NOMBRES NON ENTIER
4
5      036E  UPDDT  =0036E
6
7      0000      . =02000
8
9      2000 31C020      LXI      SP,020C0
10     2003 010080      LXI      B,08000      ;(B)=80 , (C)=00
11     2006 213320      LXI      H,TABLE      ;(HL)=ADRESSE DU HAUT
12                                     ;DE LA TABLE
13     2009 1EAB"        MVI      E,0AB      ;"AB"=NBRE A CONVERTIR
14     200B 7B          DEBUT:  MOV      A,E
15     200C BE          CMP      M          ;"AB" COMPARE AU NBRE
16                                     ;POINTE PAR (HL)
17     200D DA2320      JC       $0          ;SI CY=1 "AB" < ((HL))
18     2010 96          SUB      M          ;SINON ON SOUSTRAIT
19     2011 C5          PUSH     B          ;ON SAUVE (BC)
20     2012 F5          PUSH     PSW       ;ET (A) ET LES FLAGS
21     2013 C1          POP      B          ;(B) = (A) ET
22                                     ;(C) = FLAGS
23     2014 79          MOV      A,C
24     2015 E610      ANI      010          ;ON TESTE AC , CARRY-
25                                     ;AUXILIAIRE
26     2017 C21E20      JNZ      $1
27     201A 78          MOV      A,B          ;SI AC=0 ON RETRANCHE
28     201B D606      SUI      006          ;06
29     201D 47          MOV      B,A          ;(B)=RESULTAT
30     201E 58          $1:  MOV      E,B
31     201F C1          POP      B          ;RAPPEL DE (BC)
32     2020 79          MOV      A,C
33     2021 80          ADD      B          ;(B) = 80,PUIS 40 ...
34     2022 4F          MOV      C,A          ;(C)=NBRE CONVERTI
35     2023 78          $0:  MOV      A,B
36     2024 B7          ORA      A
37     2025 CA2E20      JZ       FIN          ;RZ SI SOUS-PROG.
38     2028 1F          RAR
39     2029 47          MOV      B,A
40     202A 2C          INR      L          ;HL POINTE LE NOMBRE
41                                     ;SUIVANT

```

```

42 202B C30B20      JMP      DEBUT
43 202E 79          FIN:    MOV      A,C
44 202F CD6E03      CALL    UPDDT
45 2032 76          HLT
46
47                  ; ((HL)) = CONTENU DE LA CASE MEMOIRE
48                  ; DONT L'ADRESSE EST (HL)
49
50 2033 50          TABLE: .BYTE    050
51 2034 25          .BYTE    025
52 2035 12          .BYTE    012
53 2036 06          .BYTE    006
54 2037 03          .BYTE    003
55 2038 02          .BYTE    002
56 2039 01          .BYTE    001
57 203A 00          .BYTE    000

```

; IL Y A 8 BOUCLES

N.B.— Le nombre traduit est dans C, B est le registre contenant 80, 40..., et E contient le nombre à traduire, diminué de 50, 25.....

```

1                  .TITLE CONVER,'HEX-DCB-DEC'
2
3                  ;CONVERSION HEXA-DCB POUR NOMBRES NON ENTIERS
4
5          036E      UPDDT      =0036E
6
7          0000      .        =02000
8
9
10 2000 31C020      LXI        SP,020C0
11 2003 011F20      LXI        B,TABLE
12 2006 1E00        MVI        E,000          ;INITIALISATION
13 2008 3EAB        MVI        A,0AB          ;"AB" EST LE NOMBRE
14                  ;A TRADUIRE
15 200A B7          ORA        A              ;CY=0 POUR NE PAS
16                  ;PERTURBER
17 200B 17          DEBUT:    RAL              ;CETTE INSTRUCTION
18 200C 57          MOV        D,A            ;SAUVE (A)
19 200D D21420      JNC        $0             ;SI CY = 0 LE BIT
20                  ;ETAIT NUL
21 2010 0A          LDAX       B              ;A CHARGE AVEC LE
22                  ;NOMBRE POINTE PAR (BC)
23 2011 83          ADD        E              ;QUE L'ON AJOUTE A "AB"
24 2012 27          DAA
25 2013 5F          MOV        E,A
26 2014 03          $0:       INX        B      ;ON AVANCE D'UN PAS
27                  ;DANS LA TABLE
28 2015 7A          MOV        A,D
29 2016 B7          ORA        A              ;POSITIONNE LES FLAGS
30                  ;NON AFFECTES PAR "MOV"
31 2017 C20B20      JNZ        DEBUT
32 201A 7B          MOV        A,E
33 201B CD6E03      CALL    UPDDT
34 201E 76          HLT

```

```

35
36 201F 50      TABLE: .BYTE 050
37 2020 25      .BYTE 025
38 2021 12      .BYTE 012
39 2022 06      .BYTE 006
40 2023 03      .BYTE 003
41 2024 02      .BYTE 002
42 2025 01      .BYTE 001
43 2026 00      .BYTE 000
44
45      0000      .END

```

On peut remplacer MVI A,X_H et MVI E, X_D dans le précédent programme par CALL RDKBD (puis MOV E, A), mais en *ne prenant pas les mêmes registres* ; ainsi, le contenu des registres sauvegardés se retrouvera dans les registres rappelés :

- PUSH PSW sauve A et F ;
- POP B range le contenu de A dans B et celui de F dans C ;
- MOV A, C met le contenu de C dans A ;
- ANI 10 teste le 4ème bit par un «ET» : si AC est nul le résultat est nul et Z = 1.

DIVISION EN HEXADECIMAL D'UN NOMBRE DE 4 CHIFFRES PAR UN NOMBRE DE 2 CHIFFRES, AVEC, VIRGULE AU RESULTAT

*But : Etude du principe de la division et son application avec une virgule au résultat.
Principales ou nouvelles instructions utilisées : SHLD, CMA, LHLD*

La division en binaire est très simplifiée par rapport au travail à effectuer en décimal puisqu'il n'y a que deux signes, 0 et 1. Donc, si le diviseur est plus petit que le dividende, on met un 1 dans le quotient, ce que montre l'exemple suivant où 7 divisé par 3 donne 2,3333... ; en binaire :

7 = 0111 ;	3 = 0011 ;	- 3 = 1101
<div style="display: flex; justify-content: space-between; width: 100%;"> 0111 0011 </div>		
<div style="display: flex; justify-content: space-between;"> <div style="width: 45%;"> <div style="text-align: right; padding-right: 10px;">1101</div> <div style="border-top: 1px solid black; padding-top: 2px;"> <div style="display: flex; align-items: center;"> <div style="border: 1px solid black; padding: 0 2px; margin-right: 5px;">1</div> 0100 > 0011 </div> </div> <div style="text-align: right; padding-right: 10px;">1101</div> <div style="border-top: 1px solid black; padding-top: 2px;"> <div style="display: flex; align-items: center;"> <div style="border: 1px solid black; padding: 0 2px; margin-right: 5px;">1</div> 0001 < 0011 </div> </div> <div style="text-align: right; padding-right: 10px;">0010 < 0011</div> <div style="text-align: right; padding-right: 10px;">0100 > 0011</div> <div style="border-top: 1px solid black; padding-top: 2px;">1101</div> <div style="border-top: 1px solid black; padding-top: 2px;"> <div style="display: flex; align-items: center;"> <div style="border: 1px solid black; padding: 0 2px; margin-right: 5px;">1</div> 0001 </div> </div> </div> <div style="width: 50%; padding-left: 10px;"> <div style="border-top: 1px solid black; padding-top: 2px;">(1 + 1) 0 1 --- = 10, 0101</div> </div> </div>		
etc.		

1	.TITLE DIVISI,'ON HEXA'				
2					
3	;DIVISION HEXADECIMALE AVEC VIRGULE				
4					
5	0363	UPDAD	=00363		
6	036E	UPDDT	=0036E		
7					
8	0000		.=02000		
9					
10	;LE RESULTAT EST EN 20B1 PARTIE BASSE DU QUOTIENT				
11	; 20B2 PARTIE HAUTE DU QUOTIENT				
12	; 20B0 PARTIE APRES LA VIRGULE				
13					
14	2000 31C020	LXI	SP,020C0		
15	2003 110000	LXI	D,00000		
16	2006 21CDAB	LXI	H,0ABCD	;"ABCD"=DIVIDENDE	
17	2009 3EEF	MVI	A,0EF	;"EF"=DIVISEUR NON NUL	
18	200B F5	PUSH	PSW	;QUE L'ON SAUVE	

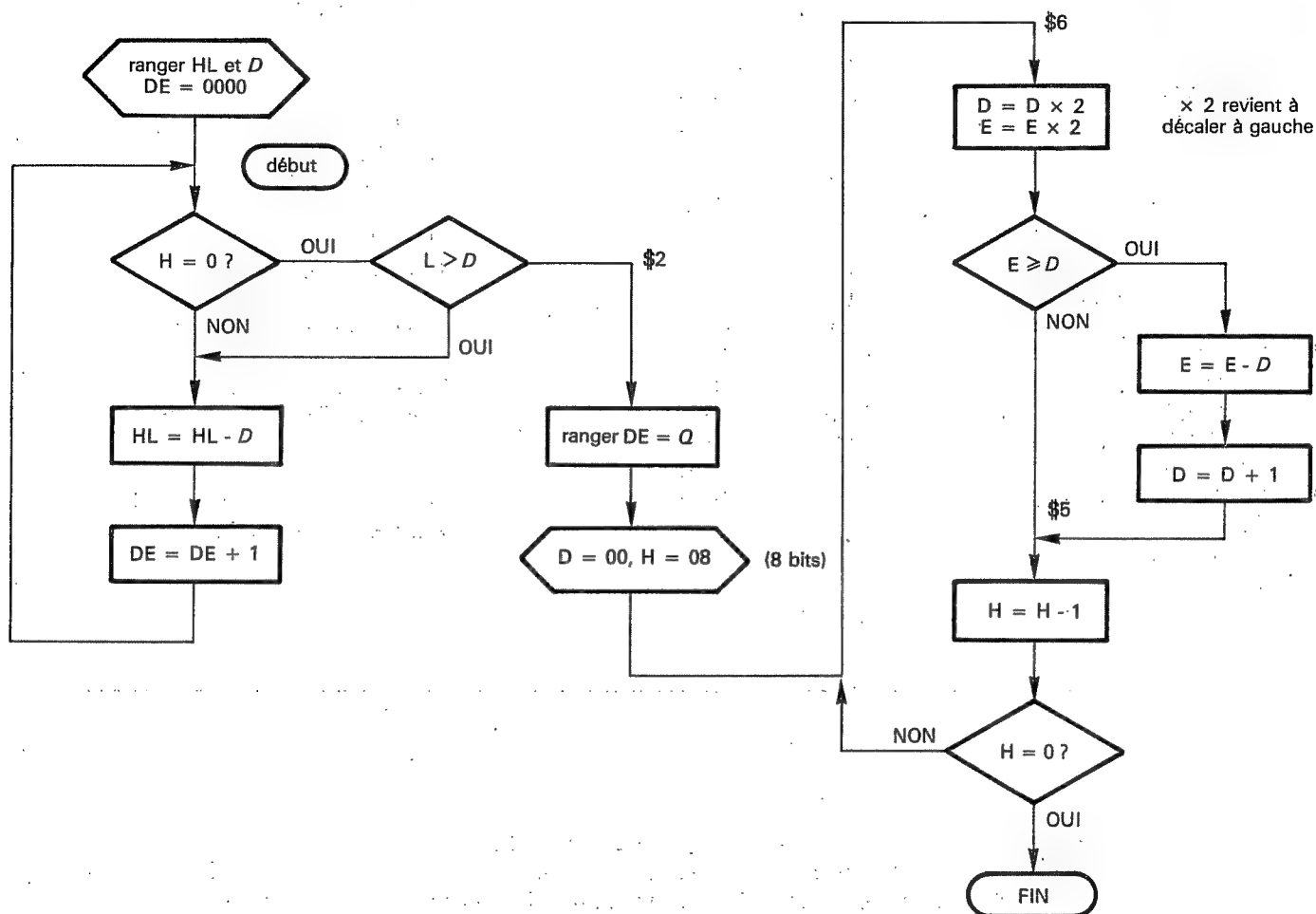
19	200C	2F		CMA		
20	200D	3C		INR	A	
21	200E	4F		MOV	C,A	;(BC) EST LE COMPLEMENT
22	200F	06FF		MVI	B,OFF	;A 2 DU DIVISEUR
23	2011	AF	DEBUT:	XRA	A	;(A)=0
24	2012	BC		CMP	H	;(H)=0?
25	2013	C21F20		JNZ	\$1	
26	2016	F1		POP	PSW	;RAPPEL DU DIVISEUR
27	2017	BD		CMP	L	;(L)>=DIVISEUR ?
28	2018	F5		PUSH	PSW	
29	2019	CA1F20		JZ	\$1	
30	201C	D22420		JNC	\$2	;(L)<DIVISEUR
31	201F	13	\$1:	INX	D	;ON INCREMENTE (DE)
32						;QUOTIENT ENTIER
33	2020	09		DAD	B	;LE DIVISEUR EST
34						;SOUSTRAIT DU DIVIDENDE
35						;(HL) = RESTE
36	2021	C31120		JMP	DEBUT	
37						;
38						;ICI (H)=00 , (L) < DIVISEUR , (DE)=QUOTIENT
39						;
40	2024	EB	\$2:	XCHG		;ECHANGE (DE) ET (HL)
41	2025	22B120		SHLD	020B1	;RANGE (L) EN 20B1
42						;ET (H) EN 20B2
43	2028	6F		MOV	L,A	;(L)=DIVISEUR
44	2029	2608		MVI	H,008	;PARTIE "DECIMALE"
45						;DE 8 BITS
46	202B	7A	\$6:	MOV	A,D	;A CONTIENT LE DERNIER
47						;QUOTIENT
48	202C	07		RLC		;ON MULTIPLIE PAR 2
49	202D	57		MOV	D,A	
50	202E	7B		MOV	A,E	;A CONTIENT LE DERNIER
51						;RESTE
52	202F	17		RAL		;CY=BIT DE POIDS FORT
53	2030	5F		MOV	E,A	
54	2031	DA3820		JC	\$4	;CY=1 , ON PEUT DIVISER
55	2034	BD		CMP	L	;(A)>=DIVISEUR ?
56	2035	DA3B20		JC	\$5	
57	2038	81	\$4:	ADD	C	;DIVISEUR RETRANCHE
58	2039	5F		MOV	E,A	
59	203A	14		INR	D	;QUOTIENT INCREMENTE
60	203B	25	\$5:	DCR	H	;ON DECREMENTE (H)
61	203C	C22B20		JNZ	\$6	
62	203F	7A		MOV	A,D	
63	2040	32B020		STA	020B0	;ON RANGE LA PARTIE
64						;APRES LA VIRGULE
65	2043	CF		RST	1	;FIN
66						
67		0000		.END		

Le programme est le suivant : l'instruction SHLD ad. range à l'adresse (ad) donnée le contenu de L et à l'adresse (ad + 1) le contenu de H ; l'instruction LHL effectue l'opération inverse. Si l'on désire afficher le résultat, on modifiera comme suit le programme.

59	203A	14	INR	D	%QUOTIENT INCREMENTE
60	203B	25	DCR	H	%ON DECREMENTE (H)
61	203C	C22B20	JNZ	%6	
62	203F	7A	MOV	A, D	
63	2040	F5	PUSH	PSW	%ON RANGE LA PARTIE
64					%APRES LA VIRGULE
65	2041	2AB120	LHLD	020B1	%ON RAPPELLE LA PARTIE
66					%ENTIERE
67	2044	EB	XCHG		%QUI EST MISE DANS DE
68	2045	0601	MVI	B, 001	%POUR AVOIR UN POINT
69	2047	CD6303	CALL	UPDAD	%AFFICHE LA PARTIE
70	204A	0600	MVI	B, 000	%ENTIERE
71	204C	F1	POP	PSW	
72	204D	CD6E03	CALL	UPDDT	%AFFICHE LA PARTIE
73	2050	76	HLT		%APRES LA VIRGULE
74					
75		0000	.END		

On peut tester la valeur de (B) en 204A par un point d'arrêt et on s'apercevra que l'instruction MVI B, 00 est inutile !

L'organigramme est donc le suivant, en choisissant de mettre le dividende dans HL, le diviseur D dans A, le quotient Q (entier) dans DE, puis la partie non entière q dans D, la partie basse du dividende étant dans E :



MULTIPLICATION EN HEXADECIMALE AVEC VIRGULE

But : Multiplier deux nombres binaires non entier, le résultat comptant une partie non entière de 8 bits.

Principales ou nouvelles instructions utilisées : Les mêmes que précédemment.

Lors de traitements mathématiques, les nombres ne sont pas toujours entiers, en particulier si l'on utilise des convertisseurs analogiques-numériques et si l'on veut une certaine précision. Le plus simple est de s'arranger pour que le produit puisse s'exprimer avec 8 bits après la virgule, c'est-à-dire pour que le bit de poids le plus faible vaille 2^{-8} de l'unité. Pour cela, l'un des nombres sera exprimé en 2^{-n} de son unité et le deuxième en 2^{n-8} de la sienne. Par exemple, la tension sera mesurée en 2^{-4} volt (1/16 de volt) et le courant en 2^{-4} ampère ; le produit sera alors en 2^{-8} watt. Ainsi, l'octet de poids le plus faible représentera la partie non entière et sera converti à l'aide du programme de conversion précédent ; de même, la partie entière pourra comporter deux octets et sera convertie elle aussi à l'aide de l'un des programmes précédents.

On obtient le programme suivant. Le multiplicateur est dans E et le multiplicande dans HL. On effectue la multiplication à l'aide de l'instruction DAD p *qui affecte la retenue (carry)* (et lui seul), ce qui permet de traiter en une seule instruction le dépassement. *Ce programme ne tient pas compte de la position de la virgule* (sauf pour l'affichage) ; c'est à l'utilisateur d'en voir les limites. Par exemple, pour :

— le multiplicateur, on a :

FF =	0,99	(1) =	0.1 1 1 1	1 1 1 1
	1,98	(2)	1.1 1 1	1 1 1 1
	3,97	(3)	1 1.1 1	1 1 1 1
	7,96	(4)	1 1 1.1	1 1 1 1
	15,93	(5)	1 1 1 1	• 1 1 1 1
	31,87	(6)	1 1 1 1	1.1 1 1
	63,75	(7)	1 1 1 1	1 1.1 1
	127,50	(8)	1 1 1 1	1 1 1.1
ou	255,00	(9)	1 1 1 1	1 1 1 1.

— et pour le multiplicande :

FFFF =	65535,00	(9)
	32767,50	(8)
	16383,75	(7)
	8191,87	(6)
	4095,93	(5)
	2047,96	(4)
	1023,97	(3)
	511,98	(2)
	255,99	(1)

Les chiffres (entre parenthèses) donnent les associations à faire. Pour exprimer le résultat en BCD, la partie entière sera limitée à $270F_H$.

```

1          .TITLE MULHEX,'VIRG'
2
3          ;MULTIPLICATION HEXADECIMALE AVEC VIRGULE
4
5          0363  UPDAD  =00363
6          036E  UPDDT  =0036E
7
8          0000          . =02000
9
10         2000 31C020      LXI      SP,020C0
11         2003 11AB00      LXI      D,000AB      ; (E)=MULTIPLICATEUR
12         2006 21EFCD      LXI      H,0CDEF      ; (HL)=MULTIPLICANDE
13         2009 E5          PUSH     H
14         200A C1          POP      B      ; (BC)=(HL)
15         200B 1D          DCR      E      ;ON DECREMENTE UNE FOIS
16                                     ;CAR LE PREMIER "DAD"
17                                     ;MULTIPLIE PAR 2
18         200C 09          $1:      DAD      B
19         200D D21120      JNC      $0      ;SI CY=0 ON PASSE
20         2010 14          INR      D      ;SINON ON INCREMENTE D
21         2011 1D          $0:      DCR      E
22         2012 C20C20      JNZ      $1      ;
23                                     ;ICI (D)="CENTAINES", (H)="DIZAINES ET UNITES"
24                                     ;ET (L)=PARTIE NON ENTIERE
25                                     ;
26         2015 5C          MOV      E,H      ;REGROUPEMENT
27         2016 7D          MOV      A,L
28         2017 F5          PUSH     PSW
29         2018 0601      MVI      B,001
30         201A CD6303      CALL     UPDAD
31         201D F1          POP      PSW
32         201E CD4E03      CALL     UPDDT
33         2021 76          HLT
34
35         0000          .END

```

TEMPORISATIONS

But : Utilisation des instructions du microprocesseur pour réaliser des temporisations.

Il est souvent nécessaire de réaliser des temporisations sur un système rapide, afin de l'adapter à un système lent. Plusieurs solutions sont possibles :

- la boucle d'attente ;
- la temporisation interne ;
- l'interruption.

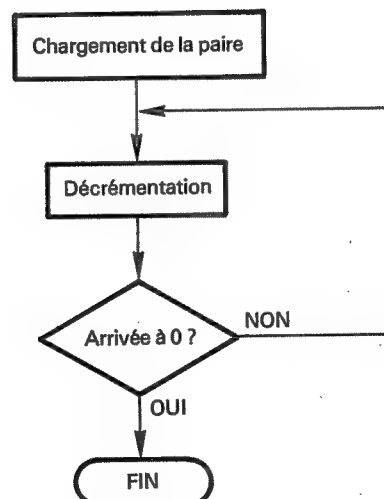
La troisième possibilité a déjà été vue : l'événement extérieur *demande* à être traité. Nous avons déjà eu un aperçu de la première dans le cas de RDKBD où l'on attend que le contenu de la cellule mémoire 20FE soit différent de 80.

Nous nous intéressons donc au deuxième cas : le microprocesseur fonctionne à un certain rythme imposé par une horloge. Pour le 8085, sur le kit, l'horloge travaille à 3 MHz environ (quartz 6,144 MHz) et les instructions demandent un certain nombre de périodes pour être exécutées :

- Toutes les opérations entre registres MOV, INR, ADD, ANA, RLC, NOP, RIM, SIM demandent 4 périodes.
- Les opérations avec la mémoire via HL, BC ou DE telles que MOV, LDAX, ADD, ANA demandent 7 périodes.
- Les opérations immédiates MVI, ADI occupent 7 périodes.
- Les opérations avec adresse LXI, MVI M, DCR M occupent 10 périodes.
- Les sauts, appels et retours demandent entre 7 et 18 périodes.

Tout cela est largement défini dans le manuel du 8085 et d'ailleurs spécifié dans l'un des tableaux donnant la liste des instructions à la fin de cet ouvrage.

Ainsi, avec un quartz à 6,144 MHz, si nous voulons obtenir une boucle durant n secondes et utilisant la décrémentation d'une paire de registres, nous devons écrire l'organigramme suivant :



Or, la *décrémentation d'une paire n'affecte pas les indicateurs* ; il faut donc passer par la *détection de l'arrivée à zéro de chaque registre composant la paire*, c'est-à-dire leur égalité simultanée à zéro ce qui est fait à l'aide de l'instruction ORA (OU logique) qui ne donne zéro que si tous les bits sont nuls. Nous aurons donc :

BOUCLE	DCX p	$p = r_1 + r_2$
	MOV A, r ₂	(r ₂) dans A
	ORA r ₁	OU de (A) donc de (r ₂) avec (r ₁)
	JNZ boucle	

La durée d'une boucle est :

DCX	=	6 périodes
MOV	=	4 périodes
ORA	=	4 périodes
JNZ	=	10 périodes
		ou 7 si on est à zéro

soit 24 périodes, la dernière passe ne comptant que 21 périodes. Si le contenu de p est N , nous aurons :

$(N - 1) \times 24 + 21$ périodes soit
 $(N \times 24 - 3)$ périodes.

Pour les temps longs (supérieurs à la milliseconde) on peut écrire $N \times 24$ périodes ce qui donne N fois $7,81 \mu s$. Pour avoir 0,5 seconde, il faut faire $N = 64020_D$, soit FA14, Pour obtenir des temps plus longs, nous réaliserons des touches emboîtées du type :

	MVI	B, N ₀
boucle 2	LXI	D, N ₁
boucle 1	DCX	D
	MOV	A, E
	ORA	D
	JNZ	boucle 1
	DCR	B
	JNZ	boucle 2

Un tel ensemble donne 256 fois $0,5 s = 128 s$ au maximum. On peut aussi mettre N_0 dans une paire :

	LXI	B, N ₀
boucle 2	LXI	D, N ₁
boucle 1	DCX	D
	MOV	A, E
	ORA	D
	JNZ	boucle 1
	DCX	B
	MOV	A, C
	ORA	B
	JNZ	boucle 2

Ce programme donne 65535 par $0,5 s = 32768 s$, soit un peu plus de 10 heures !

— *Mais le microprocesseur ne fait que cela !*. Il est donc préférable de confier ce travail à un périphérique qui prévendra le microprocesseur lorsque le temps est écoulé (par interruption). Un tel périphérique est sur le kit : il s'agit de la partie «Timer» de la RAM 8155, qui sert dans le sous-programme assurant la gestion du fonctionnement en pas à pas. La sortie Timer est liée à l'entrée «TRAP» du microprocesseur. Le Timer de la RAM travaille sous 14 bits ; il existe un circuit, le 8253 comportant 3 décompteurs travaillant sur 16 bits, mais qu'il faudra ajouter au kit.

CODAGE DES TOUCHES D'UN CLAVIER

But : Utiliser les codes affectés aux touches d'un clavier.

Principales ou nouvelles instructions utilisées : PCHL.

Nous avons vu, au cours du programme d'entrée des données au clavier, que chaque touche est codée en binaire. Ce codage est facile à obtenir puisque les touches sont des intersections d'un réseau comptant 8 colonnes et pouvant comporter 8 rangées. Le code d'une touche est alors.

0	0	N° de rangée	N° de colonne
---	---	--------------	---------------

Les numéros *rangés* et *colonne* vont de 000 à 111 et la fonction de la touche n'est définie qu'en fonction du programme au cours duquel elle est lue. Ainsi, les touches de 3 à F servent à appeler les registres si on utilise le programme EXAM-REG.

Comment utiliser le code d'une touche pour appeler un programme particulier ?

— Premier cas : trois «fonctions» au plus

Si nous n'avons à traiter que trois possibilités, au maximum, nous utilisons l'instruction CPI *data* (revoir le programme d'entrée des données au clavier). Ainsi si F₁, F₂, F₃ sont appelés par les touches codées 10, 11, 12, nous ferons après entrée du code de la touche :

```
CPI 11
JZ F2
JC F1
```

Programme pour F3

— Deuxième cas : plus de trois «fonctions»

Dans le cas où nous désirons avoir plus de trois «fonctions», il est nécessaire d'exécuter plusieurs comparaisons, ce qui devient vite prohibitif. Il est préférable d'utiliser l'instruction PCHL qui «force» dans le compteur ordinal (PC) pointant les instructions la valeur contenue dans HL. On obtient alors le morceau de programme suivant :

CDE702	CALL	RDKBD	; (OU AUTRE) (A)=CODE
			; DE LA TOUCHE
07	RLC		; (A) = (A)*2
218020	LXI	H, TABL	; RDKBD DETRUIT (HL)
85	ADD	L	; (A) = (A) + (L)
6F	MOV	L, A	; ((HL)) = ADRESSE BASSE
			; DU DEBUT DU PROGRAMME
5E	MOV	E, M	; QUI EST RANGEE DANS E
2C	INR	L	
56	MOV	D, M	; (D) = ADRESSE HAUTE
EB	XCHG		; L'ADRESSE DU PROGRAMME
			; EST DANS HL
E9	PCHL		; SAUT A CE PROGRAMME

Il faut évidemment écrire une *table des adresses* :

TABLE 1

Adresse Basse de F_0
 Adresse Haute de F_0
 Adresse Basse de F_1
 Adresse Haute de F_1
 Adresse Basse de F_2
 Adresse Haute de F_2
 Etc.

Ainsi, si « F_0 » = 10 (code de la touche appelant le programme F_0), il faut que «TABLE 0» mis dans HL soit égal à (TABLE 1 — 20_H), ce qui veut dire que TABLE 1 = 20A0 donne, avec F_0 = 10, TABLE 0 = 2080. Les codes correspondant à F_1 , F_2 , F_3 sont évidemment 11, 12, 13...
 A titre d'exemple nous allons attribuer à A, B, C, D, E... les rôles suivants :

A = affichage de 00
 B = affichage de 01
 C = affichage de 02
 D = affichage de 03
 E = affichage de 04
 Etc

Mais ces touches peuvent devenir \times , \div , $+$, $-$ si on le désire et les adresses auxquelles le programme «sautera» seront celles des programmes exécutant la multiplication, la division, l'addition et la soustraction.

```

1      .TITLE CODETO, 'UCHE'
2
3      ;APPEL D'UN PROGRAMME PAR TOUCHE CODEE
4
5      .MACRO SIM
6      .BYTE 030
7      .ENDM
8
9      02E7 RDKBD  =002E7
10     2080 TABL  =02080
11
12     0000      . =02000
13
14     2000 31C020 LXI    SP, 020C0
15     2003 3E08 MVI    A, 008
16     2005 SIM
17     2006 FB EI
18     2007 CDE702 CALL   RDKBD      ; (OU AUTRE) (A)=CODE
19                                     ; DE LA TOUCHE
20     200A 07 RLC      ; (A) = (A)*2
21     200B 218020 LXI    H, TABL    ; RDKBD DETRUIT (HL)
22     200E 85 ADD     L      ; (A)=(A)+(L)
23     200F 6F MOV     L, A      ; ((HL))=ADRESSE BASSE
24                                     ; DU DEBUT DU PROGRAMME
25     2010 5E MOV     E, M      ; QUI EST RANGEE DANS E
26     2011 2C INR     L
27     2012 56 MOV     D, M      ; (D)=ADRESSE HAUTE
28     2013 EB XCHG    ; L'ADRESSE DU PROGRAMME
29                                     ; EST DANS HL
30     2014 E9 PCHL      ; SAUT A CE PROGRAMME

```

```

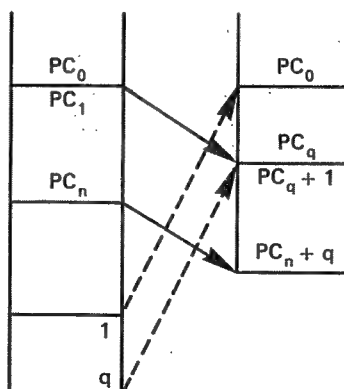
31
32 2015 3E80  PROG0:  MVI    A,080
33 2017 320019      STA    01900
34 201A 3E0E      MVI    A,00E      ;0E=0
35 201C 320018      STA    01800
36 201F 76        HLT
37 2020 3E80  PROG1:  MVI    A,080
38 2022 320019      STA    01900
39 2025 3E9F      MVI    A,09F      ;9F=1
40 2027 320018      STA    01800
41 202A 76        HLT
42 202B 3E80  PROG2:  MVI    A,080
43 202D 320019      STA    01900
44 2030 3E4A      MVI    A,04A      ;4A=2
45 2032 320018      STA    01800
46 2035 76        HLT
47 2036 3E80  PROG3:  MVI    A,080
48 2038 320019      STA    01900
49 203B 3E0C      MVI    A,00C      ;0C=3
50 203D 320018      STA    01800
51 2040 76        HLT
52          ;
53          ;
54          ;
55          ;
56          ;
57          ;
58
59
60
61
62 2041          . =02094
63
64 2094 1520  TABLE 1: .DBYTE  PROG0
65 2096 2020          .DBYTE  PROG1
66 2098 2B20          .DBYTE  PROG2
67 209A 3620          .DBYTE  PROG3
68
69          ; ICI A AFFICHE 0, B AFFICHE 1, .....
70
71          0000          .END

```


INSERTION D'UN COMPLEMENT AU PROGRAMME

Il est parfois nécessaire d'ajouter quelques instructions au sein d'un programme déjà écrit en mémoire. Comme il est fastidieux de tout devoir «ré-entrer», on fait parfois un «saut» au morceau que l'on ajoute, celui-ci se terminant par un «saut» de retour vers le programme principal. Mais, il est possible d'insérer ce morceau à l'intérieur du programme. Il faudra toutefois reprendre les adresses de sauts (et appels) éventuels.

La partie à insérer sera écrite bien *au-delà* de la fin du programme résultant, de façon à laisser la place pour le décalage, selon le schéma ci-dessous :



Il faut commencer par traduire le morceau de PC_1 à PC_n avant de pouvoir insérer les nouvelles instructions. Le programme est le suivant (les adresses ne sont pas indiquées) ; il comporte 27 octets.

1		.TITLE	INSERT	
2				
3		; INSERTION D'UN COMPLEMENT DE PROGRAMME		
4				
5	AAAA	PCO	=0AAAA	
6	BBBB	PCN	=0BBBB	
7	CC00	Q000	=0CC00	
8				
9	0000		.=00000	
10				
11	0000	01AAAA	"	LXI B,PCO ;ADRESSE DERNIER OCTET
12				;DE LA PARTIE HAUTE DU
13				;PROGRAMME
14	0003	11BBBB	"	LXI D,PCN ;ADRESSE DERNIER OCTET
15				;DU PROGRAMME INITIAL
16	0006	2100CC	"	LXI H,Q000 ;NBRE D'OCTETS A INSERER
17	0009	19		DAD D ;(HL)=ADRESSE DU DERNIER
18				;OCTET PROGRAMME FINAL
19	000A	1A	\$0:	LDAX D ;DERNIER OCTET DANS A
20	000B	77		MOV M,A ;TRANSFERE EN PCN+Q
21	000C	AF		XRA A ;ET REMPLACE PAR
22	000D	12		STAX D ; 00

23	000E	1B	DCX	D	
24	000F	2B	DCX	H	
25	0010	79	MOV	A,C	
26	0011	AB	XRA	E	; (E)=ADRESSE BASSE DU
27					; DERNIER OCTET DU PROG.
28	0012	C20A00	JNZ	\$0	
29	0015	78	MOV	A,B	; OUI , (D)=ADRESSE HAUTE
30	0016	AA	XRA	D	; DU DERNIER OCTET ?
31	0017	C20A00	JNZ	\$0	
32	001A	CF	RST	1	
33					
34		0000	.END		

La place est libre pour entrer les instructions supplémentaires. On notera que ce procédé est souvent appelé «verrue», ou «patch» par les Anglo-Saxons.

CHARGEMENT DE TABLES

But : Apprendre à travailler avec des tableaux de données, et ici à les charger.

Nombreuses sont les applications qui recourent à des tables : applications de gestion, de conversion, etc. Avant d'apprendre comment on accède à des tables pour les lire, on va examiner comment on les charge.

Pour cela, on va passer par un petit programme très simple puisque le tableau de données qu'on va dresser n'en comporte que 5, sur un octet, logées en mémoire à partir de l'adresse 2040.

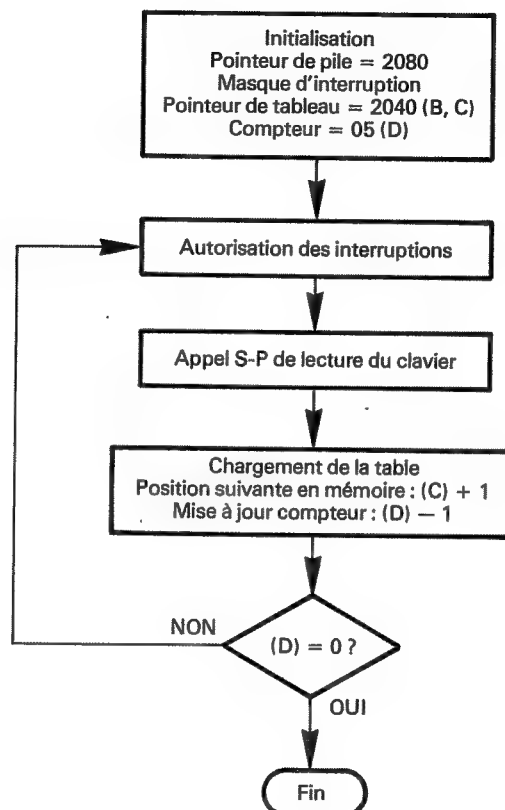
Avant toute chose, il convient d'attribuer à des registres deux fonctions essentielles, de la façon suivante :

— Le «pointeur» indique l'adresse de la prochaine cellule mémoire à charger. Il sera donc initialisé, au départ, à 2040 dans notre cas. Cette tâche de pointeur est confiée à la paire B, C.

— Le «compteur» tient à jour le compte des informations chargées. Au départ, il est initialisé à 5. On utilisera le registre D.

D'où proviennent les données ? Elles seront frappées au clavier, avec cinq frappes successives par conséquent, en faisant appel au programme moniteur de lecture du clavier stocké en 02E7. *Remarquons que ce sous-programme ne touche pas aux contenus de B, C et D, raison pour laquelle ces registres ont été choisis.*

Après avoir introduit le programme en mémoire et frappé 5 touches consécutivement, on pourra vérifier que les valeurs correspondantes ont bien été chargées dans les cellules qui leur ont été affectées.



```

1  $MOD85
2
3      NAME      CHARTAB01
4
5  ;CHARGEMENT D'UNE TABLE
6
02E7      7      RDKBD      EQU      02E7H
8
2000      9      ORG      2000H
10
2000 31C020      11 DEBUT: LXI      SP, 20C0H
2003 3E08      12      MVI      A, 0BH
2005 30      13      SIM                      ;MASQUE D'INTERRUPTIONS
2006 014020      14      LXI      B, 2040H ;POINTEUR DE TABLE
2009 1605      15      MVI      D, 05H ;COMPTEUR
200B FB      16 SUITE: EI
200C CDE702      17      CALL     RDKBD ;FRAPPE
200F 02      18      STAX     B ;DES 5
2010 0C      19      INR      C ;DONNEES ET
2011 15      20      DCR      D ;MISE A JOUR
2012 C20B20      21      JNZ     SUITE ;FINI ?
2015 CF      22      RST      1 ;OUI
23
2000      24      END DEBUT

```

Chargement sur deux digits : table des carrés

Dans le programme précédent, on n'a chargé qu'un seul digit par cellule, ce qui peut sembler un gaspillage... puisque un digit est codé sur un quartet et qu'une cellule mémoire en contient deux. En fait, seul le quartet de faible poids a été utilisé.

Pour occuper totalement les 8 bits, et par conséquent charger deux frappes consécutives dans une cellule, on frappe une première valeur, par exemple 5 ce qui donne en binaire 0101, mot qui va dans l'accumulateur en faible poids, soit :

0000 0101

On va décaler ce mot à gauche de 4 positions afin de transférer en *fort poids* le quartet de faible poids ; on obtient dans (A), avec 4 RLC (rotations) :

0101 0000

Puis; on va ranger (A) provisoirement dans un registre disponible, E. Après quoi, on peut appeler une seconde frappe au clavier, par exemple 9 en décimal, qui vient dans (A) :

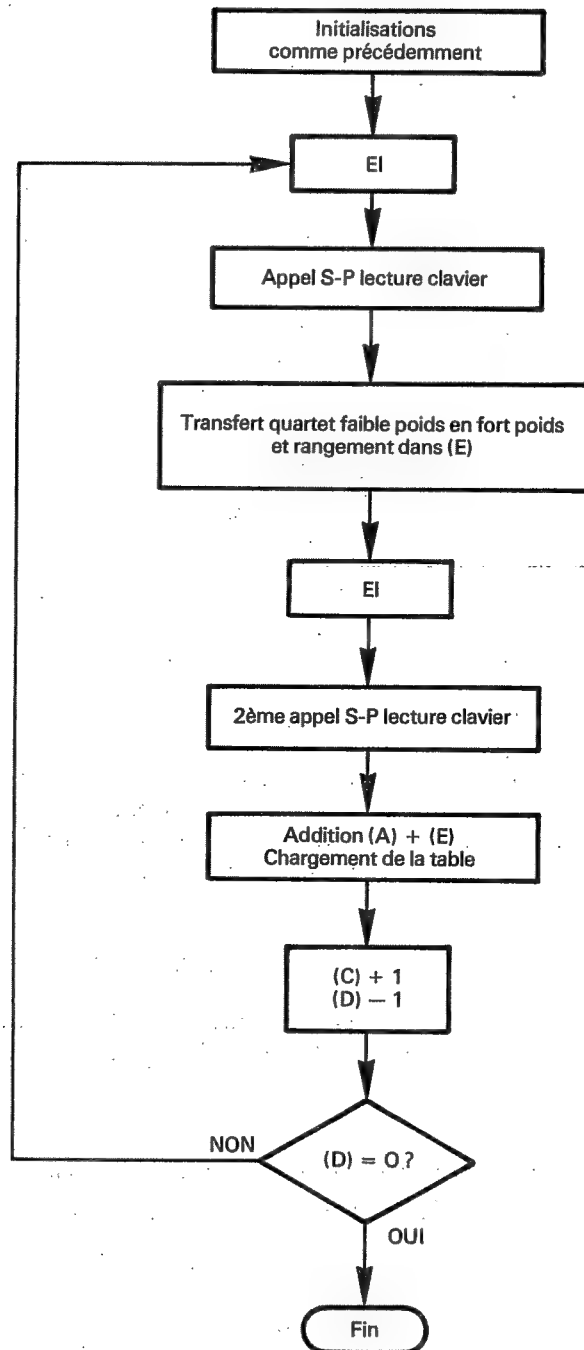
0000 1001

Si l'on fait l'addition de (E) et de (A), on réunit les deux quartets car on obtient alors dans A :

0101 1001

Cet octet traduit bien les deux frappes hexadécimales 59, et il pourra être expédié dans le tableau.

Sur ce principe, on va charger la table des carrés des 10 premiers chiffres à partir de l'adresse 2040. On logera donc successivement 00, 01, 04, 09, 16, 25, 36, 49, 64, et 81 dans dix cellules consécutives. Le programme dérive du précédent, on devra occuper 10 cellules mémoires, on va charger 0A dans (D) qui en tient le compte.



On constate que la frappe «en aveugle», sans affichage, est frustrante. On peut parfaitement procéder à un affichage en appelant les sous-programmes du moniteur affichant à droite, ou à gauche du réseau d'afficheurs ; malheureusement, ce sous-programme détruit les contenus des registres qu'il faudra, soit préserver dans la pile, soit ranger en mémoire. C'est là un exercice de programmation auquel vous êtes conviés.

LOC	OBJ	LINE	SOURCE STATEMENT
		1	\$MOD85
		2	
		3	NAME CHARTAB02
		4	
		5	;CHARGEMENT DE TABLE -NOMBRES DE 2 DIGITS
		6	
02E7		7	RDKBD EQU 02E7H
		8	
2000		9	ORG 2000H
		10	
2000 31C020		11	DEBUT: LXI SP, 20C0H
2003 3E08		12	MVI A, 08H ;COMME
2005 30		13	SIM ;PRECEDEMENT
2006 014020		14	LXI B, 2040H
2009 160A		15	MVI D, 10D
200B FB		16	SUITE: EI
200C CDE702		17	CALL RDKBD
200F 07		18	RLC ;POUR FORMER L'OCTET
2010 07		19	RLC
2011 07		20	RLC
2012 07		21	RLC
2013 5F		22	MOV E, A ;RANGEMENT DU QUARTET
2014 FB		23	EI
2015 CDE702		24	CALL RDKBD ;SECOND QUARTET
2018 83		25	ADD E ;OCTET COMPLET
2019 02		26	STAX B
201A 0C		27	INR C
201B 15		28	DCR D
201C C20B20		29	JNZ SUITE
201F CF		30	RST 1
		31	
2000		32	END DEBUT

ADRESSAGE DE TABLES PAR CALCUL D'ADRESSE

But : Apprendre à adresser des tables avec calcul d'adresse

Supposons qu'on ait rangé en mémoire, dans des cellules consécutives et à partir de l'adresse 2040, les valeurs des carrés des nombres de 0 à 9. L'occupation mémoire est donc la suivante :

(2040) =	0
(2041) =	1
(2042) =	4
(2043) =	9
(2044) =	16
(2045) =	25
(2046) =	36
(2047) =	49
(2048) =	64
(2049) =	81

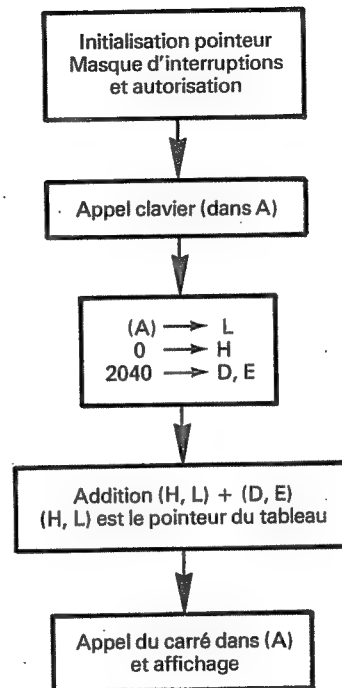
On va rédiger le programme suivant, permettant, après avoir frappé au clavier un nombre entre 0 et 9, d'afficher son carré. Pour cela, on va se servir d'un *pointeur* de tableau, la paire H, L, et on confiera l'adresse de *base* du tableau à D, E (donc 2040).

Ce type de programme convient au traitement de nombreux problèmes : calcul de fonctions mathématiques (carrés, cubes, racines...), de fonctions trigonométriques (si l'on accepte de réduire la précision, pour simplifier), et en industriel, correction de courbes de capteurs, par exemple ; ou encore, conversions de codes, etc.

Comment se fait le calcul d'adresse donnant la cellule-mémoire intéressée ? En additionnant la *base* du tableau à un *déplacement* indiqué par le pointeur. Ce déplacement est tout bonnement, ici, le nombre frappé car c'était la solution de loin la plus simple, qu'on logera dans (H, L).

Il convient donc d'effectuer une addition et l'on a choisi *une addition entre paires de registres*, ce que le 8085 permet très facilement et directement avec une seule instruction DAD, spécifiant en outre la paire visée (D, E) puisque l'autre terme de l'addition est obligatoirement contenu dans (H, L). Le résultat de l'addition allant dans HL, on pourra appeler le carré dans (A) à l'aide d'un simple MOV.

Ne pas oublier de charger au préalable les carrés en mémoire, soit directement, soit à l'aide du programme de l'exercice de chargement de tables.



Organigramme du problème de recherche en tableau

LOC	OBJ	LINE	SOURCE STATEMENT
		1	\$MOD85
		2	
		3	NAME ADDTABLE01
		4	
		5	;ADRESSAGE EN TABLE PAR CACUL D'ADRESSE
		6	
02E7		7	RDKBD EQU 02E7H
036E		8	UPDDT EQU 036EH
		9	
		10	;CHARGER LES CARRES EN 2040H ET LA SUITE
		11	
2000		12	ORG 2000H
		13	
2000	31C020	14	DEBUT: LXI SP, 20C0H
2003	3E08	15	MVI A, 08H
2005	30	16	SIM
2006	FB	17	BOUCLE: EI ;MASQUE D'INTERRUPTIONS
2007	CDE702	18	CALL RDKBD
200A	6F	19	MOV L, A ;CHARGEMENT
200B	2600	20	MVI H, 00H ;DE L'EMPLACEMENT
200D	114020	21	LXI D, 2040H ;ADRESSE DE LA BASE
2010	19	22	DAD D ;MISE A JOUR DU POINTEUR
2011	7E	23	MOV A, M ;APPEL DU CARRE
2012	CD6E03	24	CALL UPDDT ;AFFICHAGE
2015	C30620	25	JMP BOUCLE
		26	
2000		27	END DEBUT

ADRESSAGE DE TABLES PAR RECHERCHE DE LA DONNEE CONVERSION : HEXADECIMAL - 7 SEGMENTS

But : Montrer comment on retrouve la ligne d'une table lorsqu'on ne peut pas calculer son adresse, celle-ci étant obtenue à la suite de comparaisons successives.

Problème : pour afficher un digit, il faut le traduire en code à 7 segments. Plutôt que de confier à l'opérateur le soin de coder le digit en 7 segments, on va le demander à l'ordinateur. En frappant sur une touche du clavier, la machine exécutera automatiquement la conversion, et on affichera le résultat (correspondant à la touche frappée).

Pour cela, il faut loger le tableau voulu en mémoire, à partir de 2040. Ce tableau comprend deux séries d'éléments : le code hexadécimal, et puis, bien sûr, son équivalent en 7 segments. Si le premier code est stocké en 2040, celui à 7 segments ira en 2041 ; le second code hexadécimal sera en 2042 et son équivalent 7 segments en 2043, etc. Par conséquent, les codes hexadécimaux (ici, de 0 à F) seront dans les cellules d'ordre pair, ceux à 7 segments, dans des cellules d'ordre impair.

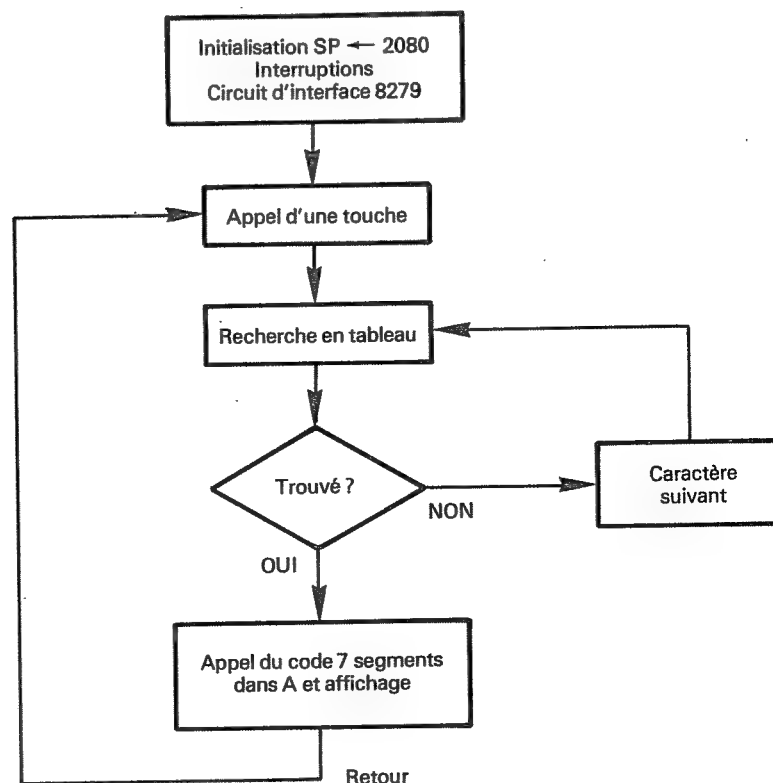
L'occupation mémoire est donc :

Adresse		Code	
Paire	Impaire	Hexa	7 segments
2040		0	
	2041		0C
2042		1	
	2043		9F
2044		2	
	2045		4A
2046		3	
	2047		0B
2048		4	
	2049		99
204A		5	
	204B		29
204C		6	
	204D		28
204E		7	
	204F		8F
2050		8	
	2051		08
2052		9	
	2053		89
2054		A	
	2055		88
2056		B	
	2057		3F
2058		C	
	2059		6C
205A		D	
	205B		1A
205C		E	
	205D		68
205E		F	
	205F		E8

Introduisez ces codes en mémoires. On va ensuite créer le programme de recherche et de conversion ; pour cela et après les inévitables initialisations, on appelle une touche qui vient dans A. On va alors comparer (A) aux contenus des cellules paires successives jusqu'à identité.

Rappelons que la comparaison consiste à soustraire le contenu de la cellule mémoire (M) de (A), donc à faire (A) - (M) ; mais ni (A) ni (M) ne sont affectés par le résultat de cette opération ; seuls, les indicateurs positionnés par le résultat (égal zéro, négatif ou non...) interviendront. Il sera donc facile, dès qu'on a décelé l'égalité (résultat nul, indicateur de zéro positionné à 1) de commander la lecture du code donné par la cellule mémoire suivant celle lue.

Le programme comprend l'initialisation du circuit d'interface 8279 commandant les afficheurs puisque, dès qu'on a obtenu le code à 7 segments, on le loge dans (A) pour l'afficher.



LOC	OBJ	LINE	SOURCE STATEMENT
		1	\$MOD85
		2	
		3	NAME ADDTABLE02
		4	
		5	;ADRESSAGE EN TABLE PAR RECHERCHE DE DONNEE
		6	
02E7		7	RDKBD EQU 02E7H
		8	
		9	;INTRODUIRE LE TABLEAU EN MEMOIRE
		10	
2000		11	ORG 2000H
		12	
2000	31C020	13	DEBUT: LXI SP, 20C0H
2003	3E08	14	MVI A, 08H
2005	30	15	SIM

2006 3E90	16	MVI	A, 90H	
2008 320019	17	STA	1900H	; INITIALISATION DU 8279
2008 FB	18 BOUCLE:	EI		
200C CDE702	19	CALL	RDKBD	; APPEL DE TOUCHE
200F 214020	20	LXI	H, 2040H	; DEBUT DE LA TABLE
2012 BE	21 NEXT:	CMP	M	; RECHERCHE
2013 CA1B20	22	JZ	BON	; DE
2016 2C	23	INR	L	; L'EQUIVALENCE
2017 2C	24	INR	L	
2018 C31220	25	JMP	NEXT	
201B 2C	26 BON:	INR	L	; APPEL DU
201C 7E	27	MOV	A, M	; CODE 7-SEGMENTS
201D 320018	28	STA	1800H	; AFFICHAGE
2020 C30B20	29	JMP	BOUCLE	
	30			
2000	31	END	DEBUT	

UTILISATION DE LA TOUCHE VECT - INTR

But : Apprendre à gérer une interruption - Etude de la pile de sauvegarde - Le problème des contacts.

Principales ou nouvelles instructions utilisées : PUSH et POP.

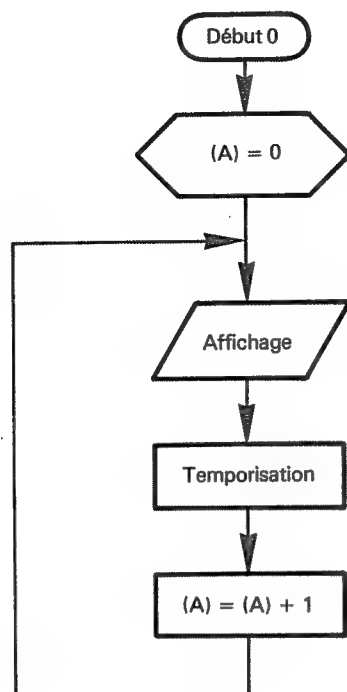
Le kit SDK 85 est prévu pour nous permettre de nous familiariser avec la gestion des interruptions en utilisant l'interruption RST 7,5.

Les interruptions réagissent, en général, à un niveau de tension, c'est-à-dire qu'il faut appliquer à l'entrée correspondante (INTR, RST 5,5; RST 6,5) une tension de 5 Volts et la maintenir jusqu'à ce que la demande soit prise en compte. Ce temps est égal, pour le 8085, à 18 périodes d'horloge (soit environ 6 μ s), c'est-à-dire à la durée de la plus longue instruction puisque l'interruption ne sera prise en compte qu'*après* la fin du traitement l'instruction au cours de laquelle elle s'est manifestée.

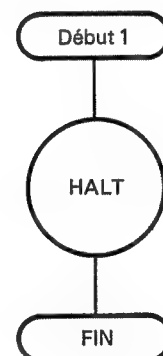
Dans le cas de l'interruption TRAP, il faut un passage de 0 à 5 Volts *et* maintien comme ci-dessus. Pour RST 7,5, il suffit d'une impulsion *positive* (passage de 0 à 5 Volts), l'information étant mémorisée dans une mémoire interne à 1 bit («flip-flop»).

Pour expérimenter, nous allons écrire un programme de comptage avec affichage, le comptage étant arrêté par l'interruption.

L'organigramme du programme principal est le suivant :



Celui du traitement de l'interruption :



Avec l'ordre HLT, le microprocesseur attend une interruption, la traite, puis repart dans le programme interrompu. Cette interruption sera évidemment différente de RST 7,5 qui nous renverrait à «début 1». Nous utiliserons RST 5,5 : une touche pressée appelle le sous-programme de traitement de RST 5.5 qui range le code de la touche en 20FE.

Programme principal :

```

13 0001          . = 02000
14
15 2000 31C020    LXI      SP, 020C0
16 2003 3E08      MVI      A, 008          ; AUTORISATION
17 2005           SIM           ; DES
18 2006 AF        XRA      A              ; (A) = 0
19 2007 FB        DEBUT:  EI           ; INTERRUPTIONS
20 2008 F5        PUSH     PSW            ; SAUVE (A) DETRUIT PAR
21                                     ; L'AFFICHAGE
22 2009 CD6E03    CALL     UPDDT
23 200C 11FFFF    LXI      D, TEMPS
24 200F CDF105    CALL     DELAI
25 2012 F1        POP      PSW
26 2013 3C        INR      A
27 2014 C30720    JMP      DEBUT

```

Note : Nous utilisons le sous-programme moniteur «délai» qui commence en 05F1 et utilise la paire DE ; au retour, nous aurons (D) = (E) = (A) = 0.

Programme d'interruption : il commence en 20CE (vérifiez la version du moniteur de votre kit) :

```

33 20CE FB        EI
34 20CF 76        HLT
35 20D0 C9        RET

```

Pour étudier les opérations effectuées lors de l'interruption nous écrirons différemment :

```

1          .TITLE VECTIN, 'TR-2'
2
3 0000          . = 020CE
4
5 20CE CF      RST      1
6 20CF FB      EI
7 20D0 C9      RET

```

Le programme une fois lancé affiche la série des nombres de 0 à FF à la cadence de 1 toutes les demi-secondes environ. L'affichage est arrêté si l'on presse la touche «VECT INTR» et repart si on presse n'importe quelle autre touche.

Que se passe-t-il exactement ? En utilisant le deuxième programme d'interruption, dès que la touche «VECT INTR» est pressée, «8085» est affiché ; on peut observer les registres et les mémoires comme on l'a expliqué dans un programme précédent. On obtient (se souvenir du nombre affiché au moment de l'arrêt) :

(SPH) = 20
(SPL) = BA

20BA = Fx adresse de retour à l'intérieur du
BB = 05 programme «délai»

BC = 12 adresse de retour après traitement du
BD = 20 programme «délai»

BE = (F) contenu du registre des indicateurs
BF = (A) nombre affiché au moment où l'on a pressé «VECT INTR»

Pour voir le processus de retour, on peut forcer DE à 0001 en pressant successivement EXEC, puis EXAM REG, puis D, en formant le premier octet 00, puis NEXT et le second octet 01, et enfin en reprenant le pas à pas avec SINGLE STEP. On entrera dans le programme «délai» à l'adresse lue de 20BA et 20BB, après être passé par 20 CF et 20 D0 ; on effectuera une boucle dans le programme «délai» et on reviendra au programme principal en 2012.

Pour analyser le fonctionnement de HLT, écrivons le programme suivant :

```

5 20CE C32020      JMP      INTR
6
7 20D1             . = 02020
8
9 2020 FB          INTR:    EI
10 2021 76          HLT
11 2022 3AFE20      LDA      020FE      ; (A) = (20FE)
12 2025 CF          RST      1
13

```

Noter qu'on ne peut examiner le contenu de 20FE après RST 1 car le programme moniteur met 80 dans cette case.

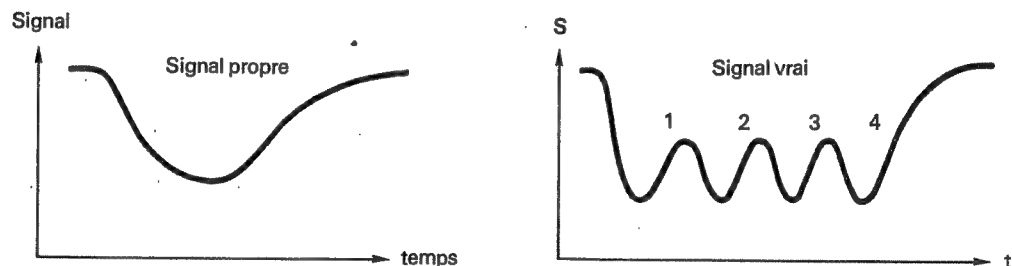
Une fois le programme lancé, on presse «VECT INTR» puis une touche X (retenir laquelle) ; le microprocesseur affiche «8085» et l'on peut examiner les registres et les mémoires :

(A)	= 0X	touche numérique ou
ou	= 1X	touche fonction
(SPH)	= 20	
(SPL)	= xy	indéterminés !!
20 xy	= 22	
20 (xy + 1)	= 20	
20B8	= 22	
20B9	= 20	
20BA	= Fu	Adresse interne
20BB	= 05	au programme «délai»
20BC	= 12	
20BD	= 20	
20BE	= (F)	
20BF	= N	= nombre affiché au moment où l'on a pressé «VECT INTR»

Nous constatons que le pointeur de pile est «remonté» très haut (on dit «remonter» en raison du dessin que nous faisons de la pile de sauvegarde), et que nous n'aurons pas deux fois la même valeur de SPL ! La touche pressée se retrouve bien dans (A).

Pourquoi alors le pointeur de pile ne pointe-t-il pas toujours la même adresse ?

La touche «VECT INTR» est un *simple contact à lamelle*. La fermeture du contact provoque la décharge d'un condensateur et l'ouverture, la charge ; c'est cette opération qui déclenche la demande d'interruption. Le système présente des «rebonds». Nous n'aurons donc pas un signal pur mais des oscillations :



Nous avons donc l'équivalent de plusieurs demandes d'interruption ; si la deuxième apparaît après que la première soit traitée, elle est prise en considération. Dans notre cas, le traitement est très rapide : EI ne demande que 4 périodes d'horloge, donc on arrive à HLT environ $6\mu s$ après la demande d'interruption ; de ce fait les demandes numéro 2, 3 et 4 seront prises en compte puisque les écarts en temps sont supérieurs à $6\mu s$, le microprocesseur ne repartant que sur interruption après un HLT. En comptant le nombre d'adresses 2022, on obtient le nombre de rebonds. Tout ceci nous incitera à nous méfier, lors de l'écriture au programme de traitement de RST 7,5, de la «montée» de la pile qui *peut détruire* une partie de notre programme. Cet inconvénient est éliminé :

- si le programme de traitement ne permet pas une nouvelle interruption ;
- s'il est suffisamment long pour qu'aucune demande supplémentaire n'apparaisse. Dans ce cas, on n'autorise les interruptions qu'après avoir remis à zéro le flip-flop de RST 7,5 par les instructions suivantes :

```
MVI A, 0001 1 x x x
SIM
EI
```

Le cinquième bit dans A, mis à 1 avant SIM, «efface» la mémoire de RST 7,5. Cette précaution n'étant pas prise, on reviendrait dans le programme de traitement de l'interruption. Ajoutons que ce flip-flop est automatiquement mis à zéro lors du traitement de l'interruption.

Remarques

1 - On trouve dans la pile de sauvegarde l'adresse 2022 qui est celle de LDA, adresse de retour après traitement de l'interruption faisant repartir le microprocesseur. En réalité, la pile est «montée» de 6 cases en plus : deux pour l'adresse de retour, après traitement de RST 5,5, et quatre pour le traitement de RST 5,5 qui comporte deux PUSH (voir programme ININT du moniteur en 028E).

2 - On peut faire un test de la remontée de la pile en remplissant de 20A0 à 20C0 de 00 puis, après affichage de «8085», on va lire le contenu de ces mémoires. La première case différente de 00 vous indique la hauteur maximale de la pile.

3 - Lors de l'appel des sous-programmes du moniteur (affichage, etc.), la pile peut monter et à la limite, déborder sur les programmes stockés en RAM qu'elle détruira. On veillera donc à ne pas dépasser les limites permises.

4 - Pour comprendre l'utilisation de SIM, on peut ordonner :

- MVI A, 0E qui interdit RST 7,5 : il n'y aura pas d'arrêt du comptage ;
 - MVI A, 09 qui interdit RST 5,5 : il n'y aura pas de redémarrage.
-

EXERCICES AVEC LE 8279

But : Apprendre à utiliser le circuit 8279 et à adresser des périphériques.

Principales ou nouvelles instructions utilisées : les mêmes que précédemment.

Affichage

Nous avons déjà vu quelques possibilités offertes par le circuit périphérique 8279 au cours d'un programme précédent. Nous avons vu que, pour afficher un caractère, il fallait d'abord *prévenir* le circuit du mode d'affichage, puis lui donner *le code* du caractère.

La notice relative à ce circuit nous indique qu'il peut gérer 16 afficheurs maximum et que l'entrée des caractères peut se faire à *droite* ou à *gauche*, ces deux modes n'étant distincts que dans le cas d'une auto-incrémentation. Or, nous ne possédons que 6 afficheurs ; aussi, les mots d'ordre seront les suivants :

- entrée à gauche 00
- entrée à droite 10
- affichage auto incrémenté 9 x (où x est une valeur indifférente).

D'où le programme-exercice suivant

```

1          .TITLE  UTILIS,'8279-1'
2
3          ;UTILISATION DU 8279 EXEMPLE 1
4
5          05F1    DELAI    =005F1
6
7          0000          . =02000
8
9          2000 31C020          LXI      SP,020C0
10         2003 2619          MVI      H,019          ;ON MET 19 DANS H
11         2005 3600          MVI      M,000          ;(MVI M,010) ON ENVOIE
12                                     ;LE PREMIER C.W. AU
13                                     ;8279 A L'ADRESSE 19**
14         2007 3693          MVI      M,093          ;DEUXIEME C.W. ORDRE
15                                     ;D'AFFICHAGE
16         2009 25          DCR      H          ;(H)=18
17         200A 369F          MVI      M,09F          ;09F= 1
18         200C 11FFFF          LXI      D,OFFF
19         200F CDF105          CALL     DELAI
20         2012 364A          MVI      M,04A          ;04A= 2
21         2014 11FFFF          LXI      D,OFFF
22         2017 CDF105          CALL     DELAI
23         201A 360B          MVI      M,00B          ;00B= 3
24         201C 11FFFF          LXI      D,OFFF
25         201F CDF105          CALL     DELAI
26         2022 3699          MVI      M,099          ;099= 4

```

27	2024	11FFFF	LXI	D, 0FFFF	
28	2027	CDF105	CALL	DELAI	
29	202A	3629	MVI	M, 029	; 029 = 5
30	202C	11FFFF	LXI	D, 0FFFF	
31	202F	CDF105	CALL	DELAI	
32	2032	3628	MVI	M, 028	; 028 = 6
33	2034	76	HLT		
34					
35		0000	.END		

Dans le cas où nous avons mis 00 en 2006, nous voyons :

E			1		
E			1	2	
E			1	2	3
E			1	2	3
E			1	2	3
6			1	2	3

Le E de gauche est le E d'exécution ; on ne voit pas 4 et 5 car il y a 6 digits (afficheurs à segments) et 8 cases mémoires prévues (le mot de commande précise 8 ou 16).
 Dans le cas où nous mettons 10 en 2006 nous voyons :

		1			X	X = indéterminé
		1	2	X	X	
	1	2	3	X	E	E = Exécution
	2	3	4	X	E	
	3	4	5	E		
	4	5	6		1	

Nous constatons que la case 0 correspond au digit le plus à gauche dans le cas d'une entrée à gauche, et le plus à droite dans le cas d'une entrée à droite ; mais les digits 1 2 3... sont dans le même ordre :

- entrée à gauche : 0 1 2 3 4 5 (6 7)
- entrée à droite : 1 2 3 4 5 6 (7 0)

Dans les deux cas, si nous envoyons un caractère alors que le précédent est dans la case 7, celui-ci est mis dans la case 0 (6 remplace E car nous sommes partis de la case 3).

En «entrée à droite», il y a décalage vers la gauche par rotation des cases. En réalité, les codes restent dans les cases mémoires (*RAM-afficheurs*) mais les numéros des cellules ne correspondent plus aux numéros des afficheurs.

A la fin du programme précédent, si nous mettons RST 1, nous ne lisons pas «8085» puisque nous ne ré-initialisons pas le circuit. Il faut, dans le cas d'une entrée à droite, passer par RST 0 (soit C7) et *sauver* (A). Le premier ordre du moniteur est :

```
0000 3E 00      MVI A,00
```

Lecture des cellules mémoires de la «RAM-afficheurs»

Les quantités que nous donnons au 8279 pour affichage sont stockées dans les 8 (ou 16) cases dans la «RAM-afficheurs» et peuvent être lues. Le mot d'ordre est 6 x ou 7 x (mode auto-incrémenté).

A titre d'exemple, nous proposons le programme suivant. Il faut remarquer que le contenu de L n'est pas utilisé si on choisi d'écrire l'adresse du périphérique dans HL, ce qui nous permet de mettre le nombre de touches ou de caractères à lire dans L.

```

3          ;UTILISATION DU 8279 EXEMPLE 2
4
5          .MACRO SIM
6          .BYTE 030
7          .ENDM
8
9          02E7  RDKBD  =002E7
10
11 0000      . =02000
12
13 2000 31C020  LXI    SP,020C0
14 2003 210719  LXI    H,01907      ;(H)=19 ADRESSE DU 8279
15                                     ;(L)=07 7 TOUCHES
16 2006 3600    MVI    M,000        ;ON PEUT NE PAS DONNER
17                                     ;CET ORDRE DONNE PAR LE
18                                     ;MONITEUR
19 2008 3690    MVI    M,090
20 200A 25      DCR    H
21 200B 3E08    MVI    A,008
22 200D        SIM
23 200E FB      EI                ;AUTORISATION DES
24 200F E5      PUSH    H          ;INTERRUPTIONS
25 2010 CDE702  CALL    RDKBD      ;SAUVE (HL) DETRUIT PAR
26 2013 E1      POP     H          ; RDKBD
27 2014 77      MOV     M,A        ;AFFICHAGE DE LA TOUCHE
28 2015 2D      DCR    L
29 2016 C20E20  JNZ     $0

```

30	2019	24	INR	H	; (H)=19
31	201A	3670	MVI	M,070	; OU 7X ,OU 6X C.W. DE
32					; LECTURE
33	201C	2E07	MVI	L,007	
34	201E	25	DCR	H	
35	201F	014020	LXI	B,02040	
36	2022	7E	MOV	A,M	; LE CONTENU DE LA CASE
37					; POINTEE PAR (HL) EST
38					; RANGE DANS LA CASE
39	2023	02	STAX	B	; POINTEE PAR (BC)
40	2024	0C	INR	C	
41	2025	2D	DCR	L	
42	2026	C22220	JNZ	\$1	
43	2029	CF	RST	1	
44					
45		0000	.END		

A la fin, quand «- 8085» est affiché, on lit dans les cases mémoires 2040 à 2046 les touches pressées dans l'ordre si on a écrit 70 en 201B, dans un ordre perturbé si on a écrit 7 x ; par contre, on lit dans toutes les cases la même touche si on a écrit 6 x. *Il faut* envoyer l'ordre de lecture juste avant la lecture. Pour en terminer avec l'affichage, il existe un *ordre d'extinction totale* (CD), mais le circuit est indisponible pendant 160 μ s. Au lieu d'attendre ce délai, on peut lire le registre d'état du circuit qui nous renseigne sur :

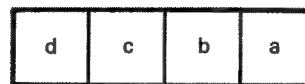
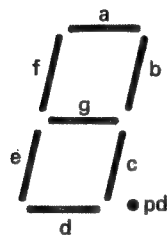
- la disponibilité de l'afficheur : bit 7 ;
- un dépassement de chargement de la «RAM CLAVIER bit 5 ;
- une lecture tentée alors que la RAM-CLAVIER est vide : bit 4 ;
- une RAM-CLAVIER pleine : bit 3 ;
- le nombre de touches pressées bits 0 à 2.

Ainsi, soit le programme ci-dessous :

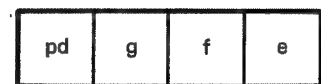
3	#UTILISATION DU 8279 EXEMPLE 3				
4					
5	05F1	DELAI	=005F1		
6					
7	0000		.=02000		
8					
9	2000	31C020	LXI	SP,020C0	
10	2003	2619	MVI	H,019	
11	2005	3680	MVI	M,080	
12	2007	25	DCR	H	
13	2009	3699	MVI	M,099	
14	200A	11FFFF	LXI	D,OFFF	
15	200D	CD F105	CALL	DELAI	
16	2010	24	INR	H	
17	2011	36CD	MVI	M,0CD	; CD= C.W. D'EXTINCTION
18	2013	7E	MOV	A,M	; LECTURE DU REGISTRE
19					; D'ETAT DU 8279
20	2014	E680	ANI	080	; BIT 7 (LE 8 EME) =1 ?
21	2016	C21320	JNZ	\$0	
22	2019	25	DCR	H	
23	201A	363E	MVI	M,03E	
24	201C	76	HLT		

Ce programme affiche 4, puis u , si l'on n'attend pas la disponibilité du circuit (on supprime la boucle d'attente), 4 disparaît mais u n'apparaît pas.

On peut interdire l'affichage (concept *différent* de l'extinction). Dans ce cas, les données en RAM sont conservées mais l'allumage des segments interdit. Le mot de commande est A3 : si l'on met A1 ou A2, seule une partie des segments est éteinte *a, b, c, d* ou *e, f, g* et *pd* (point, ou virgule décimale). Selon le tableau rappelé ci-dessous, car il a déjà été présenté :



A
(éteint par A1)



B
(éteint par A2)

On peut modifier un caractère en n'en modifiant qu'une partie et en s'interdisant de toucher à l'autre par A8 ou A4 (on ne modifie que B ou A). Cette dernière possibilité ne nous semble utile que si les sorties du 8279 servent à autre chose qu'à un affichage 7 segments. A l'aide de «A3», on peut obtenir un affichage clignotant :

```

3          ;UTILISATION DU 8279 EXEMPLE 4
4
5      05F1    DELAI    =005F1
6
7      0000          . =02000
8
9      2000 31C020          LXI      SP,020C0
10     2003 2619          MVI      H,019
11     2005 3690          MVI      M,090
12     2007 25          DCR      H
13     2008 3629          MVI      M,029
14     200A 360C          MVI      M,00C
15     200C 3629          MVI      M,029
16     200E 24          INR      H
17     200F 36A0    $0:    MVI      M,0A0          ;ALLUMAGE
18     2011 11FFFF          LXI      D,0FFFF
19     2014 CDF105          CALL     DELAI
20     2017 36A3          MVI      M,0A3          ;EXTINCTION
21     2019 11FFFF          LXI      D,0FFFF
22     201C CDF105          CALL     DELAI
23     201F C30F20          JMP      $0

```

Entrée clavier

Au lieu d'utiliser le sous-programme moniteur RDKBD, on peut écrire un programme comportant une boucle d'attente dans laquelle on lit le registre d'état attendant que le bit 0 soit à 1 ; à la sortie de la boucle, on lit le code de la touche à l'aide du mot d'ordre 40 (lecture «RAM-CLAVIER»).

```

3          ;UTILISATION DU 8279 EXEMPLE 5
4
5          036E   UPDDT   =0036E
6
7 0000          . =02000
8
9 2000 31C020    LXI     SP,020C0
10 2003 2619     MVI     H,019
11 2005 7E       $0:    MOV     A,M
12 2006 E601     ANI     001
13 2008 0A0520   JZ      $0
14 200B 3640     MVI     M,040
15 200D 25       DCR     H
16 200E 7E       MOV     A,M
17 200F CD6E03   CALL    UPDDT
18 2012 76       HLT

```

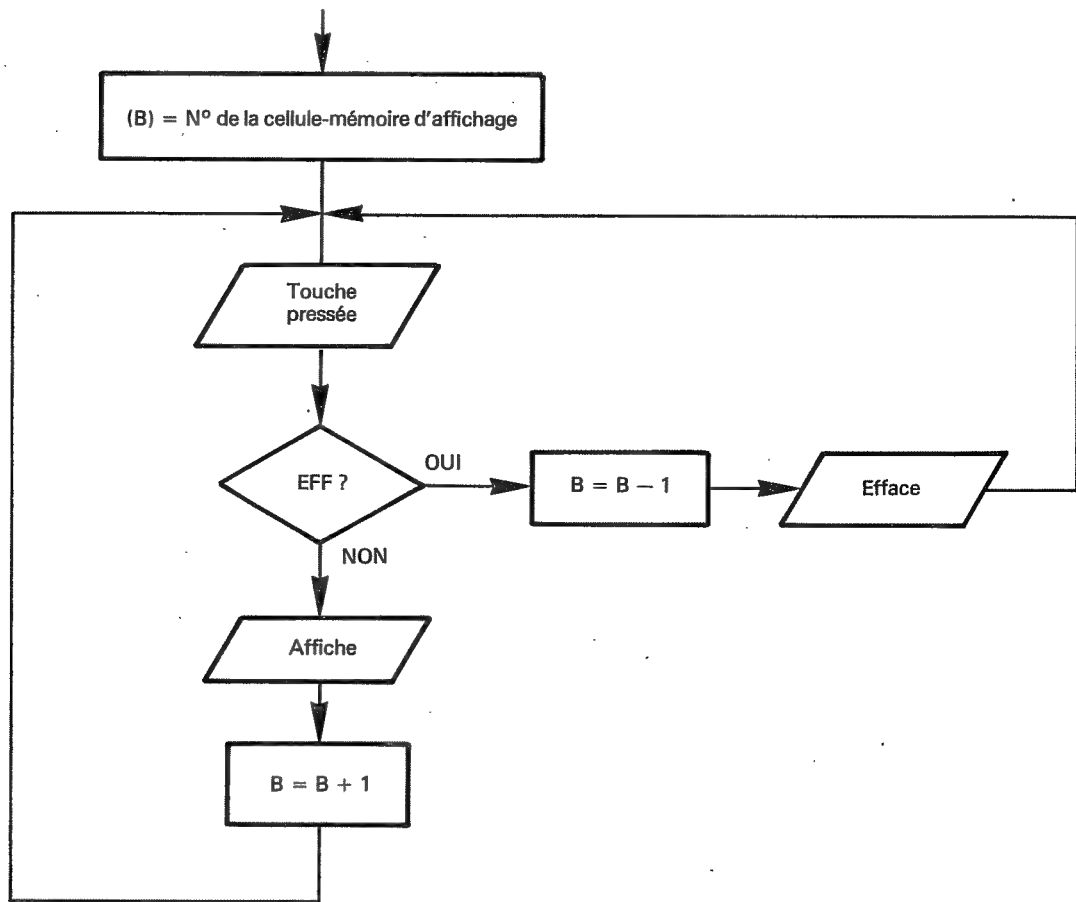
On rencontre les possibilités suivantes : en 2007, on a écrit 01 ; dès qu'une touche est pressée, on lit le contenu de la RAM-CLAVIER et on l'affiche. Si on a écrit 08, il faut attendre que la RAM-CLAVIER soit pleine pour qu'on lise une case mémoire qui est toujours la première.

AFFICHAGE SEQUENTIEL AVEC EFFACEMENT PUIS ENTREE EN MEMOIRE

But : Utilisation plus complète du 8279.

Principales ou nouvelles instructions utilisées : Les mêmes que précédemment.

Le programme proposé permet d'écrire un «texte» de 6 caractères avec correction des erreurs et entrée du texte en mémoire. Le petit problème à résoudre est l'effacement. On ne travaille pas en affichage auto-incrémenté et par conséquent, on indique à chaque fois l'emplacement du caractère selon l'organigramme partiel suivant :



La septième touche différente de «effacement» provoque l'entrée en RAM :

```

1          .TITLE  AFFSEQ
2
3          ;AFFICHAGE SEQUENTIEL AVEC CORRECTION ET ENTREE MEMOIRE
4
5          .MACRO  SIM
6          .BYTE  030
7          .ENDM
8
9          02E7  RDKBD  =002E7
10
11 0000          . =02000
12
13 2000 31C020    LXI      SP,020C0
14 2003 210719    LXI      H,01907          ; (H)=19
15                                     ; (L)=7 CARACTERES
16 2006 3600      MVI      M,000          ;ENTREE A GAUCHE
17                                     ;8 AFFICHEURS
18 2008 0680      MVI      B,080          ;PREMIERE CASE RAM-AFF.
19 200A 3E08      MVI      A,008
20 200C          SIM
21 200D FB        BCLE1: EI
22 200E 70        MOV      M,B
23 200F E5        PUSH     H              ; (H) DETRUIT PAR RDKBD
24 2010 CDE702    CALL     RDKBD
25 2013 E1        POP      H
26 2014 FE10      CPI      010          ;10=EXEC.
27 2016 CA3120    JZ        EFF
28 2019 25        DCR      H              ; (H)=18
29 201A 77        MOV      M,A          ;AFFICHAGE
30 201B 24        INR      H              ; (H)=19
31 201C 04        INR      B              ; (B) = (B) +1
32 201D 2D        DCR      L
33 201E C20D20    JNZ      BCLE1
34 2021 3670      BCLE2: MVI      M,070          ;C.W. LECTURE RAM-AFF.
35 2023 2E07      MVI      L,007
36 2025 015020    LXI      B,02050          ;TABLE DE STOCKAGE
37 2028 25        DCR      H
38 2029 7E        MOV      A,M          ;TRANSFERT DE RAM-AFF.
39 202A 02        STAX     B              ;EN RAM-STOCK.
40 202B 0C        INR      C
41 202C 2D        DCR      L
42 202D C22120    JNZ      BCLE2
43 2030 CF        RST      1
44
45 2031 05        EFF:   DCR      B
46 2032 70        MOV      M,B
47 2033 25        DCR      H
48 2034 36FF      MVI      M,0FF          ;EFFACEMENT
49 2036 24        INR      H
50 2037 2C        INR      L              ;NOMBRE DE TOUCHES
51 2038 C30D20    JMP      BCLE1
52
53          0000          .END

```

Dans ce cas, l'affichage ne signifie pas grand chose mais ce procédé est à utiliser avec un affichage alphanumérique via une mémoire décodeuse.

CHENILLARD (journal lumineux)

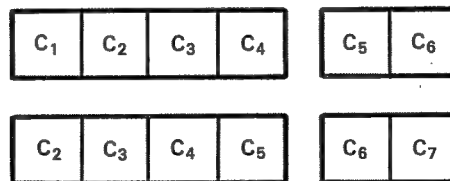
But : Utilisation du 8279 et de l'affichage.

Principales ou nouvelles instructions utilisées : Les mêmes que précédemment.

Pour réaliser un chenillard avec notre kit, nous avons deux solutions.

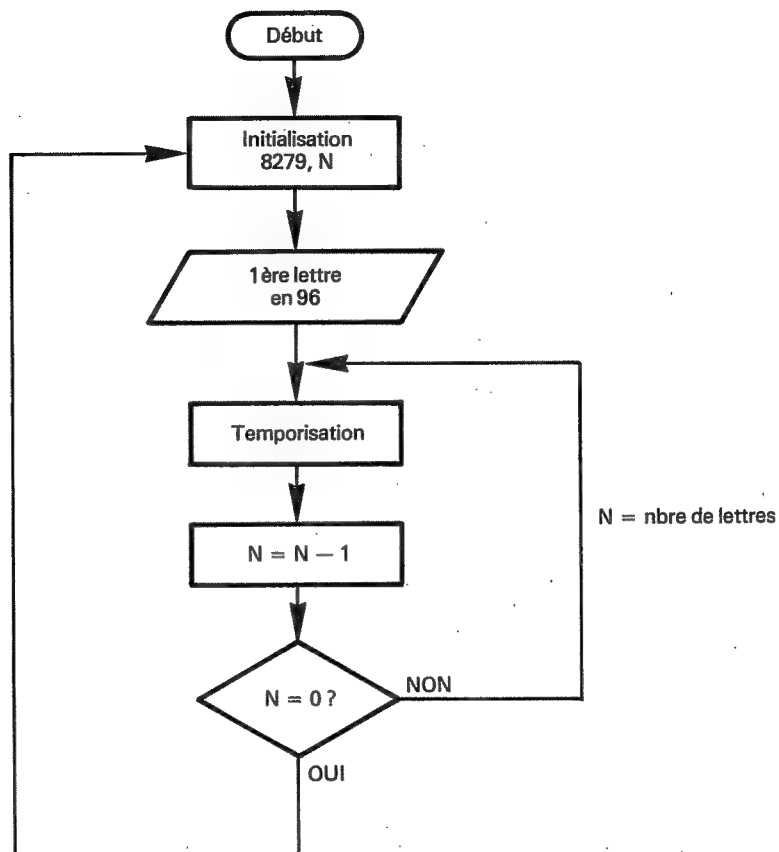
— *Première solution* : afficher 6 caractères pendant un certain temps, puis 6 nouveaux caractères avec un décalage d'un caractère avec les précédents, ce qui donne sur les afficheurs.

puis



etc..

— *Deuxième solution* : on utilise l'entrée à droite. On envoie les caractères un à un avec un certain rythme. C'est cette dernière solution que nous donnons.



On range le nombre de lettres du texte dans L puisque L n'est pas utilisé par l'adressage du 8279, ainsi que nous l'avons vu... le programme sera donc le suivant :

```

1          .TITLE CHENIL, 'LARD'
2
3          ;CHENILLARD DU JOURNAL LUMINEUX
4
5          05F1    DELAI    =005F1
6
7          0000          .#=02000
8
9          2000 31C020          LXI      SP,020C0
10         2003 21AB19 DEBUT:  LXI      H,019AB          ; (L)="AB"=NOMBRE DE
11                                     ;CARACTERES
12         2006 3610          MVI      M,010          ;ENTREE A DROITE
13         2008 3696          MVI      M,096          ;POSITION DE LA PREMIERE
14                                     ;LETTRE
15         200A 012020          LXI      B,02020          ;ADRESSE DE LA PREMIERE
16                                     ;LETTRE
17         200D 25          DCR      H
18         200E 0A          $0:      LDAX  B          ;CARACTERE DANS A
19         200F 77          MOV      M,A          ;CARACTERE AFFICHE
20         2010 11FFFF          LXI      D,0FFFF          ;TEMPORISATION
21         2013 CDF105          CALL    DELAI
22         2016 0C          INR      C
23         2017 2D          DCR      L
24         2018 C20E20          JNZ     $0
25         201B C30320          JMP     DEBUT
26
27
28         ;EXEMPLE "AB"=30
29         ;2020 FA BF 68 BA FF BA 68 FF 29 68 FA F8 FF 1A 68 FF
30         ;2030 7A 3A 3E FA BF FA FF BF 9F FF E8 88 3E F8 FF C8
31         ;2040 88 FA F8 BF FA FF 88 FF C8 3A BF BA F8 FF FF FF
32         ;
33         ;CETTE DEVISE EST EGALEMENT VRAIE POUR LES MICRO.
34
35         0000          .END

```

ENTREES-SORTIES DU KIT SDK 85 : CARREFOUR

But : Apprendre à utiliser les entrées-sorties du kit.

Principales ou nouvelles instructions utilisées : IN, OUT, RIM.

Entrée-sortie séries

Le microprocesseur 8085 possède une entrée et une sortie série (SID et SOD), ce qui permet de recevoir ou d'émettre une information unique ou plusieurs informations qui se suivent (*entrée-sortie série*). Ainsi, en début du programme moniteur, on va lire l'état de l'entrée série. Si elle est à 0 (0 volt), on travaille avec le clavier ; si elle est à 1 (environ 5 V), on travaille avec un téléimprimeur.

L'état de l'entrée série est donné par l'instruction RIM qui met dans l'accumulateur le mot suivant (où M signifie «masque d'interruption») :

Etat de l'entrée SID	I 7,5	I 6,5	I 5,5	IE	M 7,5	M 6,5	M 5,5
----------------------	-------	-------	-------	----	-------	-------	-------

La position IE est à 1 si des interruptions sont autorisées, à 0 dans le cas contraire (il n'y a pas eu autorisation, ou une interruption est/ou a été traitée). La position I indique, elle, si l'interruption est en attente de traitement.

Pour lire l'état de l'entrée série, on peut faire un ORA A qui positionne les indicateurs, puis un test sur le *bit de signe* ou un RLC et un test sur le carry (retenue).

Pour sortir une information sur la *sortie série*, on utilise l'instruction SIM déjà rencontrée au cours du programme d'entrée des données au clavier. On charge au préalable l'accumulateur avec le mot suivant (x est un état indifférent) :

Etat à mettre en sortie	Autorisation de sortie	x	reset 7,5	M	M 7,5	M 6,5	M 5,5
-------------------------	------------------------	---	-----------	---	-------	-------	-------

Ainsi :

```
MVI A, CX
SIM
```

met la sortie série à 1, et :

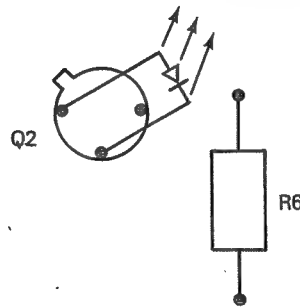
```
MVI A, 4X
SIM
```

la met là 0, alors que :

```
MVI A, 8X ou MVI A, 0X
SIM
```

sont sans action.

A titre d'exemple, si vous n'avez pas câblé l'adaptation téléimprimeur vous soudez à la place de Q₂ (transistor 2907) une diode électroluminescente (LED) comme suit :



On mettra en R₆ une résistance légèrement inférieur à 1 k Ω . Vous ferez clignoter la LED (si elle est correctement câblée, elle s'allume à la mise sous tension, sinon la «retourner») avec le programme suivant :

```

9 0000          .*=02000
10
11 2000 31C020    LXI      SP,020C0
12 2003 3E40      BOUCLE: MVI      A,040
13 2005          SIM              ;J'ETEINS*
14 2006 11FFFF    LXI      D,0FFFF
15 2009 CDF105    CALL     DELAI
16 200C 3EC0      MVI      A,0C0
17 200E          SIM              ;J'ALLUME
18 200F 11FFFF    LXI      D,0FFFF
19 2012 CDF105    CALL     DELAI
20 2015 C30320    JMP      BOUCLE
21
22          ;* UN 1 ETEINT LA LED CAR SON ANODE EST A 5 VOLTS

```

Un 1 éteint la LED car son anode est à 5 V. Pour un autre exemple d'utilisation de SID et SOD, on se reportera à l'appendice du «User's Manual» concernant l'adaptation à un magnétophone à cassettes.

Entrées-Sorties parallèles

Les mémoires qui composent le kit ont la particularité d'être des mémoires comportant en plus des entrées-sorties. Elles incluent, en effet, dans le même boîtier, des circuits de liaison parallèle avec l'extérieur (huit informations sortent ou entrent simultanément par «port» d'accès).

La mémoire 8755 (ou 8355) compte deux de ces ports ; la RAM en compte deux de 8 bits et un de 6 bits. Avant d'entrer ou de sortir une information, il faut *prévenir* les circuits du sens dans lequel ils vont être utilisés ; il faut donc envoyer un *mot de commande*. Les entrées-sorties étant sur le même boîtier que la mémoire, elles auront le même «*chip-select*» (CS : *sélection du circuit*), c'est-à-dire que l'adresse des «ports» est constituée de l'adresse haute de la mémoire et de l'adresse interne du port.

Ainsi, pour la ROM d'adresse haute 0000XXX, les adresses sont :

- port A = 00
- port B = 01
- mot de commande de A = 02
- mot de commande de B = 03

Pour la RAM d'adresse haute 00100XXX, les adresses sont :

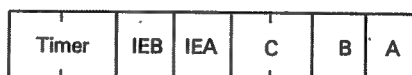
- port A = 21
- port B = 22
- port C = 23
- mot de commande = 20
- timer = 24 et 25

Il faut un mot de commande par port de la ROM car ils sont à bits indépendants : un même port peut avoir des fils d'entrée et des fils de sortie. Le mot de commande est :



- Un 1 signifie que le fil est une sortie.
- Un 0 signifie que le fil est une entrée.

Pour les RAM, il y a un mot de commande pour les trois ports et le timer :



Comme précédemment, un 0 signifie que le port est un port d'entrée. Pour C, il faut deux bits car ce port peut servir à gérer A, ou A et B ; ainsi, nous avons les possibilités :

- | | |
|----|--|
| 00 | le port C est un port d'entrée |
| 11 | le port C est un port de sortie |
| 01 | le port C gère le port A et est un port de sortie sur 3 bits |
| 10 | le port C gère les ports A et B |

Cette gestion fonctionne sur le mode «interruption», c'est-à-dire que le périphérique connecter au port A (ou B) demande par l'intermédiaire de C à recevoir ou émettre une donnée* (pour plus de détail, voir le *User's Manual*). Les deux bits réservés au timer assurent son démarrage et son arrêt, immédiat ou différé.

Pour apprendre à travailler avec les ports, on peut connecter des LEDS entre un port et le + 5 V à travers des résistances (comme pour la sortie série) et réaliser ainsi une signalisation de carrefour.

Adressage des ports

Nous avons vu que les adresses des ports et des mots de commande sont écrites sur un octet. Pour sortir le contenu de l'accumulateur ou mettre dans l'accumulateur l'état des fils d'un port, on utilise les ordres OUT et IN suivis de l'adresse du port concerné.

Ainsi :

3E	01	MVI	A, 01
D3	20	OUT	20

met le port A de la RAM en port de sortie et les ports B et C en ports d'entrée. D'autre part :

DB	22	IN	22
----	----	----	----

met dans l'accumulateur l'état du port B ; et enfin :

3E	XY	MVI	A, XY
D3	21	OUT	21

sort sur les fils du port A le mot XY.

* C'est pourquoi on trouve IEB et IEA (*Interrupt Enable...*)

Carrefour

A titre d'exemple, nous écrirons un programme assurant la gestion d'un carrefour dont l'état normal est la succession des feux rouge, vert, orange et l'état, commandé par interruption, l'orange clignotant. Les fils du port sont codés comme suit :

A ₀	Vert	1
A ₁	Orange	1
A ₂	Rouge	1
A ₃	Vert	2
A ₄	Orange	2
A ₅	Rouge	2

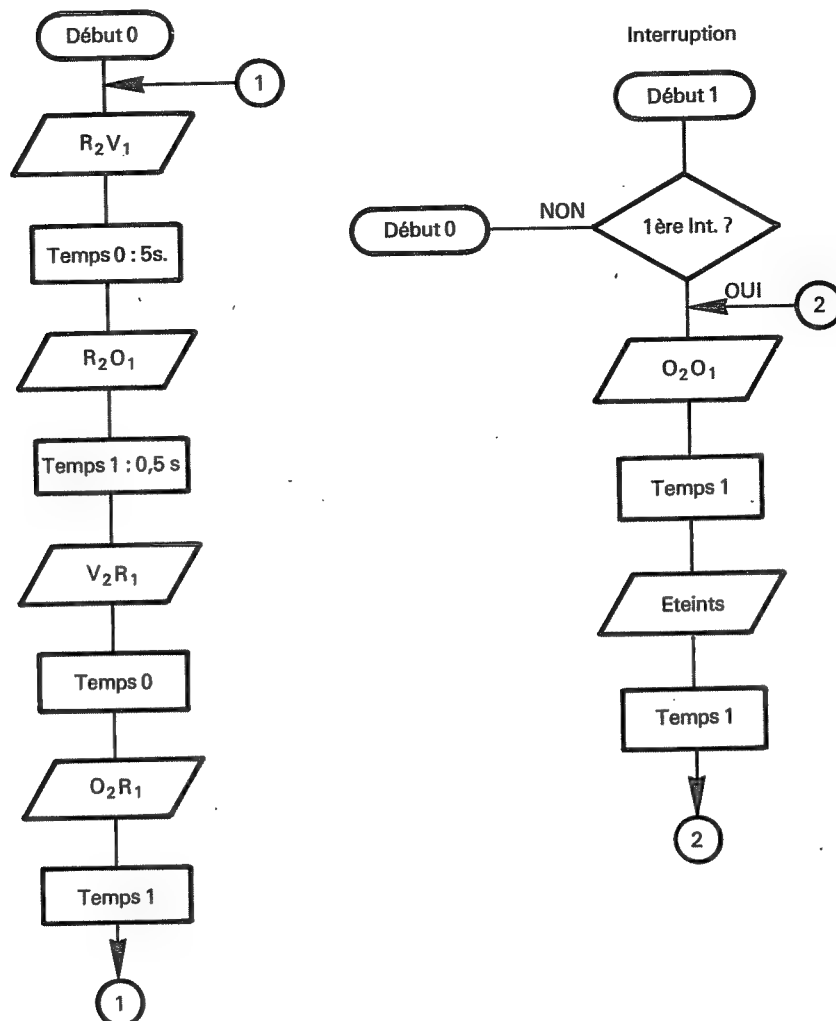
soit :

X	X	R ₂	O ₂	V ₂	R ₁	O ₁	V ₁
---	---	----------------	----------------	----------------	----------------	----------------	----------------

Un 0 allume la LED correspondante, c'est-à-dire que :

R ₂ V ₁	est donné par	1E
R ₂ O ₁	est donné par	1D
V ₂ R ₁	est donné par	33
O ₂ R ₁	est donné par	2B
O ₂ Q ₁	est donné par	2D

L'organigramme est le suivant :



```

1          .TITLE   CARREF,'OUR'
2
3          ;FEUX DE CARREFOUR AVEC CHANGEMENT DE CYCLE PAR INTR.
4
5          .MACRO   SIM
6          .BYTE   030
7          .ENDM
8
9          05F1     DELAI   =005F1
10         001E     R2V1    =01E
11         001D     R201    =01D
12         0033     V2R1    =033
13         002B     O2R1    =02B
14         002D     O201    =02D
15
16 0000          . =02000
17
18 2000 31C020 DEBUT: LXI      SP,020C0
19 2003 3E18      MVI      A,018          ;MASQUE D'INTR. AVEC
20                                     ;R.A.Z. DEMANDE V.I.
21 2005          SIM
22 2006 FB        EI
23 2007 3E01      MVI      A,001          ;C.W. LE PORT A EST
24 2009 D320      OUT      020          ;UN PORT DE SORTIE
25 200B 32FF20    STA      020FF        ;POUR LE PAS A PAS
26                                     ;AVEC SDK 85
27 200E 47        MOV      B,A          ;(B)=01 POUR COMPTER
28                                     ;LES INTERRUPTIONS
29 200F 3E1E      BOUCLE: MVI      A,R2V1
30 2011 D321      OUT      021
31 2013 CD2E20    CALL     TEMP00
32 2016 3E1D      MVI      A,R201
33 2018 D321      OUT      021
34 201A CD3820    CALL     TEMP01
35 201D 3E33      MVI      A,V2R1
36 201F D321      OUT      021
37 2021 CD2E20    CALL     TEMP00
38 2024 3E2B      MVI      A,O2R1
39 2026 D321      OUT      021
40 2028 CD3820    CALL     TEMP01
41 202B C30F20    JMP      BOUCLE
42
43 202E 0E0A      TEMP00: MVI      C,00A
44 2030 CD3820    BOUCL1: CALL     TEMP01
45 2033 0D        DCR      C
46 2034 C23020    JNZ      BOUCL1
47 2037 C9        RET
48
49 2038 11FFFF    TEMP01: LXI      D,0FFFF
50 203B CDF105    CALL     DELAI
51 203E C9        RET
52

```

53	203F	05	INTR:	DCR	B	;(B)=0 :1ERE INTERRUPT.
54						;(B)≠0 :ON CLIGNOTE SINON
55	2040	C20020		JNZ	DEBUT	;(B)≠0 :ON CLIGNOTE SINON
56	2043	3E2D	BOUCL2:	MVI	A,0201	;(B)≠0 :ON CLIGNOTE SINON
57	2045	D321		OUT	021	;(B)≠0 :ON CLIGNOTE SINON
58	2047	CD3820		CALL	TEMPO1	;(B)≠0 :ON CLIGNOTE SINON
59	204A	3EFF		MVI	A,OFF	;(B)≠0 :ON CLIGNOTE SINON
60	204C	D321		OUT	021	;(B)≠0 :ON CLIGNOTE SINON
61	204E	CD3820		CALL	TEMPO1	;(B)≠0 :ON CLIGNOTE SINON
62	2051	3E18		MVI	A,018	;(B)≠0 :ON CLIGNOTE SINON
63	2053			SIM		;(B)≠0 :ON CLIGNOTE SINON
64	2054	FB		EI		;(B)≠0 :ON CLIGNOTE SINON
65	2055	C34320		JMP	BOUCL2	;(B)≠0 :ON CLIGNOTE SINON
66						;(B)≠0 :ON CLIGNOTE SINON
67	2058			.	=020CE	;(B)≠0 :ON CLIGNOTE SINON
68						;(B)≠0 :ON CLIGNOTE SINON
69	20CE	C33F20		JMP	INTR	;(B)≠0 :ON CLIGNOTE SINON
70						;(B)≠0 :ON CLIGNOTE SINON
71		0000		.	END	;(B)≠0 :ON CLIGNOTE SINON

A la deuxième interruption, on reprend le programme principal ; il est plus facile de revenir à «début», cela évite de se poser des problèmes pour la pile.

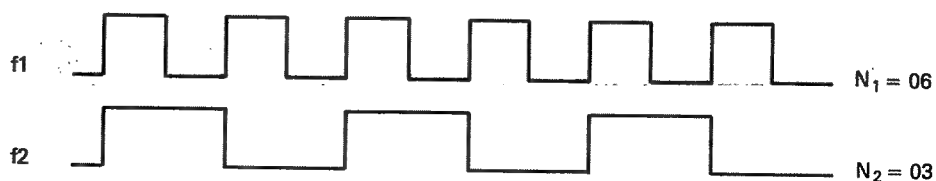
GENERATION D'UNE NOTE

But : Création d'une note à fréquence et durée déterminées

Si nous désirons transformer notre kit en piano ou en boîte à musique, il faudra nous intéresser à ce qui caractérise une note, c'est-à-dire sa *fréquence* et sa *durée*.

Nous ne pouvons, à l'aide seulement des circuits disponibles sur le kit, générer des signaux sinusoïdaux, mais seulement des signaux carrés. Or, on apprend en mathématique qu'un signal carré se compose de signaux sinusoïdaux dont la fréquence est égale à celle du signal carré multipliée par un nombre impair. Les amplitudes des «harmoniques» décroissent comme l'inverse des nombres impairs. Avec un signal carré, on obtient donc à l'oreille une fréquence accompagnée de ses harmoniques impairs.

Comment générer un signal carré de durée donnée ? Un signal carré est caractérisé par un état *haut* et un état *bas* de même durée qui est la demi-période du signal (la période est l'inverse de la fréquence). Une durée de note sera donc fixée par un nombre de demi-périodes ; donc, si deux notes de fréquences différentes ont la même durée, les nombres de périodes ne seront pas les mêmes. La figure ci-dessous donne l'exemple pour des fréquences différant d'un facteur 2 (octave)

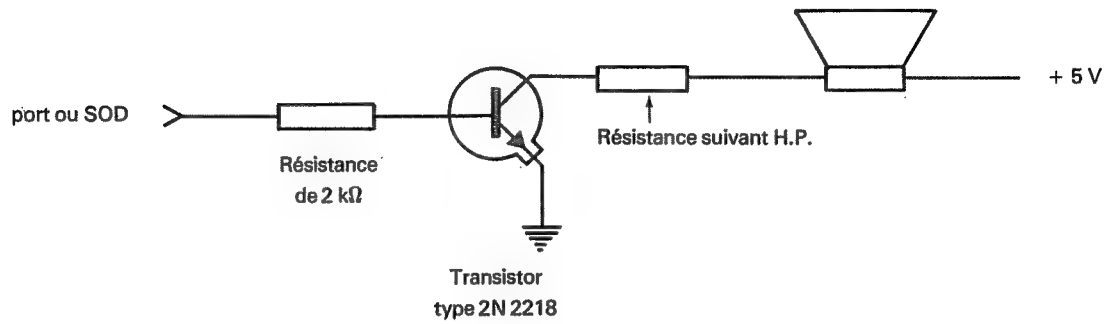


La note f 2 est plus grave que la note f 1

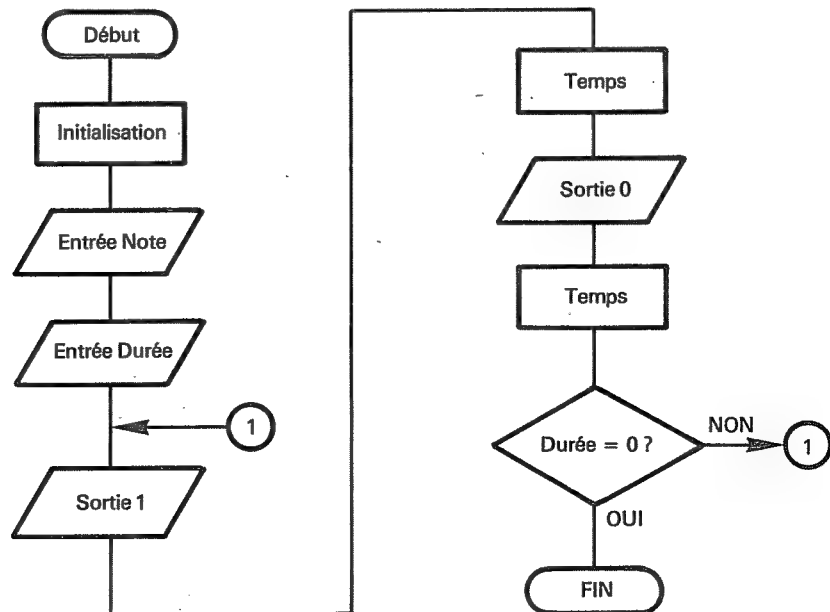
La fréquence du *La*³ de la gamme a été fixée à 440 Hz, c'est-à-dire que la demi période est de 1,14 ms (millisecondes). Pour obtenir une telle durée en décrémentant un registre (comme on l'a fait dans un programme précédent), il faut charger ce dernier à 250 (soit FA en hexa décimal). Nous voyons que nous serons obligés de prendre une paire de registres pour générer des notes plus graves, à moins que nous écrivions un programme qui double le nombre attaché à la note. Dans ce cas, nous obtenons le tableau suivant où la colonne *durée* correspond à une «durée» arbitraire qu'il faudra multiplier par un nombre adéquat pour obtenir le «rythme»

Note	Fréquence (Hz)	Nombres correspondants	
		à la note	à sa durée
do ⁴	523	68	5F
si	494	6E	5A
^b si	466	75	55
la ³	440	7C	50
[#] sol	415	84	4B
sol	392	8B	48
[#] fa	370	93	44
fa	349	9C	40
mi	329	A5	3C
^b mi	312	AE	39
ré	294	B9	36
[#] do	277	C4	33
do ³	261.5	D0	30

Il faudra réaliser un montage dont le schéma est donné ci-dessous :



et écrire un programme dont l'organigramme est le suivant :



Programme avec sortie par le port A de la RAM :

```

1          .TITLE  NOTE
2
3          ;GENERATION D'UNE NOTE MUSICALE
4
5          0010  RYTHME  =010
6          007C  LA3     =07C
7          0050  DUREE   =050
8
9          0000          . =02000
10
11 2000 31C020 DEBUT: LXI    SP,020C0
12 2003 3E01      MVI    A,001          ;PORT A = SORTIE
13 2005 D320      OUT    020
14 2007 32FF20    STA    020FF
15 200A 1610      MVI    D,RYTHME
16 200C 0E7C      MVI    C,LA3          ;7C=LA3
  
```

```

17 200E 0650      BOUCL1: MVI      B,DUREE
18 2010 3EFF      BOUCL0: MVI      A,OFF
19 2012 D321              OUT      021          ;SORTIE D'UN 1
20 2014 CD2520      CALL     NOTE
21 2017 D321              OUT      021          ;SORTIE D'UN 0
22 2019 CD2520      CALL     NOTE
23 201C 05              DCR      B
24 201D C21020      JNZ      BOUCL0
25 2020 15              DCR      D
26 2021 C20E20      JNZ      BOUCL1
27 2024 CF              RST      1
28
29 2025 1E02      NOTE:  MVI      E,002          ;FREQUENCE "NORMALE"
30 2027 79              BOUCL3: MOV      A,C
31 2028 3D              BOUCL2: DCR      A
32 2029 C22820      JNZ      BOUCL2
33 202C 1D              DCR      E
34 202D C22720      JNZ      BOUCL3
35 2030 C9              RET
36
37      0000              .END

```

Ce programme vous donne le La^3 pendant 3 secondes environ. On augmentera (ou diminuera) la durée de la note en modifiant le contenu de D (200B) ; on changera de gamme en changeant le contenu de E (2026) ; avec (E) = 01, on obtient le La^4 (880 Hz) ; (E) = 04 donne le La^2 (220 Hz) et (E) = 03 donne le $Ré^3$ (294 Hz).

REALISATION D'UN «PIANO»

But : Développement du programme précédent et application «appels» du clavier.

On peut attribuer à chaque touche une note ; nous avons donc 22 notes, en utilisant les touches-fonctions avec *Exec* = 10, *Next* = 11, *GO* = 12, *Subst-Mem* = 13, *Exam-Reg* = 14, *Single Step* = 15. La durée d'une note est fixée par programme mais le circuit 8279 comportant un mémoire-clavier de huit cases, on pourra presser plusieurs fois une touche ce qui aura pour effet d'augmenter la durée de la note correspondante.

Le programme sera une synthèse du programme antérieur «d'appel clavier» et de l'exécution d'une note du programme précédent. Il faudra écrire *deux* tables ; l'une attribuera à chaque touche une note, l'autre donnera pour chaque note sa durée. On obtient alors le programme suivant :

#TABLE NOTES				#TABLE DUREES			
0000	.=02000			2016	.=02020		
2000 F8	.BYTE	0F8		2020 28	.BYTE	028	
2001 E9	.BYTE	0E9		2021 2B	.BYTE	02B	
2002 DC	.BYTE	0DC		2022 2D	.BYTE	02D	
2003 D0	.BYTE	0D0	#D03	2023 30	.BYTE	030	
2004 C4	.BYTE	0C4		2024 33	.BYTE	033	
2005 B9	.BYTE	0B9		2025 36	.BYTE	036	
2006 AE	.BYTE	0AE		2026 39	.BYTE	039	
2007 A5	.BYTE	0A5		2027 3C	.BYTE	03C	
2008 9C	.BYTE	09C		2028 40	.BYTE	040	
2009 93	.BYTE	093		2029 44	.BYTE	044	
200A 8B	.BYTE	08B		202A 48	.BYTE	048	
200B 84	.BYTE	084		202B 4B	.BYTE	04B	
200C 7C	.BYTE	07C		202C 50	.BYTE	050	
200D 75	.BYTE	075		202D 55	.BYTE	055	
200E 6E	.BYTE	06E		202E 5A	.BYTE	05A	
200F 68	.BYTE	068		202F 5F	.BYTE	05F	
2010 62	.BYTE	062		2030 65	.BYTE	065	
2011 5C	.BYTE	05C		2031 6B	.BYTE	06B	
2012 57	.BYTE	057		2032 72	.BYTE	072	
2013 52	.BYTE	052		2033 78	.BYTE	078	
2014 4E	.BYTE	04E		2034 80	.BYTE	080	
2015 49	.BYTE	049	#FA4	2035 87	.BYTE	087	

```

64
65
66 2036 31C020      LXI      SP,020C0
67 2039 3E08        MVI      A,008
68 203B             SIM
69 203C 3E01        MVI      A,001
70 203E D320        OUT      020      ;PORT RAM=SORTIE
71 2040 32FF20      STA      020FF
72 2043 FB          $0:      EI
73 2044 CDE702      CALL     RDKBD      ;APPEL CLAVIER
74 2047 6F          MOV      L,A        ;(H)=20 PAR RDKBD
75 2048 46          MOV      B,M        ;(B)=NOTE
76 2049 C620        ADI      020
77 204B 6F          MOV      L,A
78 204C 4E          MOV      C,M        ;(C)=DUREE
79 204D 3EFF        MVI      A,0FF
80 204F D321        OUT      021
81 2051 CD6120      CALL     NOTE
82 2054 AF          XRA      A
83 2055 D321        OUT      021
84 2057 CD6120      CALL     NOTE
85 205A 0D          DCR      C
86 205B C24D20      JNZ      $1
87 205E C34320      JMP      $0        ;NOTE SUIVANTE
88
89 2061 1E02        NOTE:    MVI      E,002      ;GAMME
90 2063 78          NOTE1:   MOV      A,B
91 2064 3D          NOTE2:   DCR      A
92 2065 C26420      JNZ      NOTE2
93 2068 1D          DCR      E
94 2069 C26320      JNZ      NOTE1
95 206C C9          RET

```

Il est à noter que ce circuit 8279 peut commander un clavier de 64 touches et que les deux bits de poids forts du code des touches sont mis à zéro par des cavaliers sur le kit (9-10 ; 11-12) qui correspondent à «SHIFT» et «CONTROL». Si vous construisez votre «piano», vous pourrez utiliser ces bits pour définir une durée plus importante de la note ou une valeur différente (256 notes possibles !!).

BOITE A MUSIQUE

But : Mieux maîtriser la programmation, sous forme d'un divertissement.

Au lieu de fournir des notes une à une au microprocesseur, on peut écrire en mémoire une suite de notes avec leur durée, ce qui nous donne une mélodie dont on pourra faire varier le rythme ou (et) la gamme. Ce programme inclut les pauses et silences. Il suffit de disposer d'un espace mémoire suffisant pour pouvoir faire jouer au microprocesseur un long morceau. Nous donnons en exemple un extrait (80 notes) de «la Truite» de Schubert. Le tableau présenté avec l'avant-dernier programme donne une durée équivalente à une blanche pour un «rythme» de 07.

```

1          .TITLE  BOITE,' A MUSIQUE'
2
3 0000          . =02000
4
5          0007    RYTHME  =007
6          0002    GAMME   =002
7          FFFF    LIMITE  =FFFF
8
9 2000 31C020    LXI      SP,020C0
10 2003 3E01     MVI      A,001
11 2005 D320     OUT      020
12 2007 32FF20   STA      020FF
13 200A 215020   DEBUT:  LXI      H,02050          ;DEBUT DES NOTES
14 200D 46       $3:     MOV     B,M              ; (B) =NOTE
15 200E 2C       INR      L
16 200F 1607     MVI      D,RYTHME
17 2011 4E       $2:     MOV     C,M              ; (C) =DUREE
18 2012 3EFF     $1:     MVI      A,OFF
19 2014 D321     OUT      021
20 2016 CD3F20   CALL     NOTE
21 2019 AF       XRA      A
22 201A D321     OUT      021
23 201C 0D       DCR      C
24 201D C21220   JNZ      $1
25 2020 15       DCR      D
26 2021 C21120   JNZ      $2
27 2024 06FF     MVI      B,OFF          ;DETACHE LES NOTES
28 2026 CD3F20   CALL     NOTE
29 2029 2C       INR      L
30 202A 3EFF     MVI      A,LIMITE
31 202C BD       CMP      L
32 202D CA3E20   JZ       FIN              ;OU DEBUT
33 2030 7E       MOV     A,M
34 2031 B7       ORA      A
35 2032 C20D20   JNZ      $3              ;EST-CE UNE FAUSE ?
36 2035 2C       INR      L              ;OUI
37 2036 46       MOV     B,M

```

```

38 2037 CD3F20          CALL    NOTE
39 203A 2C              INR      L
40 203B C31120          JMP      $2
41 203E CF              FIN:    RST      1
42
43 203F 1E02      NOTE:    MVI      E,GAMME
44 2041 7B      NOTE1:    MOV      A,B
45 2042 3D      NOTE2:    DCR      A
46 2043 C24220          JNZ      NOTE2
47 2046 1D              DCR      E
48 2047 C24120          JNZ      NOTE1
49 204A C9              RET
50
51
52      ; NOTES DE LA TRUITE (GAMME=2 ,RYTHME=7,DEBUT=2050)
53
54
55      $9C 10 75 16 75 16 5C 1B 5C 1B 75 2B 9C 10 9C 10 9C 1B
56      $9C 0B 6B 0C 75 0B 7C 0A 8B 09 9C 20 00 40 9C 10 75 16
57      $75 16 5C 1B 5C 1B 75 2B 9C 10 75 16 7C 14 8B 09 7C 0A
58      $75 16 A5 0F 9C 20 00 40 9C 10 7C 14 7C 14 75 0B 7C 0A
59      $8B 09 7C 0A 75 2B 9C 10 75 16 7C 14 7C 14 7C 0A 57 0F
60      $6B 0C 7C 0A 75 2B 00 80 75 16 8B 12 8B 12 8B 12 75 16
61      $75 16 9C 10 9C 10 9C 1B 9C 0B 6B 1B 7C 14 75 2B 00 40
62      $75 16 7C 14 8B 12 8B 09 75 0B 7C 0A 6B 0C 75 2B 9C 10
63      $9C 10 9C 1B 9C 0B 6B 1B 7C 14 75 2B 00 80 9C 20
64
65      0000          .END

```

CARILLON DE PORTE

A l'aide du programme précédent et d'une analyse de touches, on peut construire un carillon de porte qui joue quelques notes d'une mélodie. Cette mélodie dépend de la touche pressée ; on peut donc savoir qui sonne à la porte !. Dans notre cas, la touche 0 nous donne «La Truite» ; 1 : «La Marseillaise» ; 2 : «L'internationale» ; 3 «Les Trois Orfèvres»...

Avant les notes, il faut donner le *ton* et le *rythme*. Chaque mélodie compte 7 notes. Si l'on désire inclure plus de notes, il faut modifier le programme comme suit :

— Mélodie de 16 octets (7 notes) : 4 RLC et ADI 10 (en 2019) ;

— Mélodie de 32 octets (15 notes) : 3 RRC et ADI 20.

N.B. : attention à la pile et à son remplissage !

Pour un système indépendant, on remplacera l'appel clavier par la séquence suivante (en branchant la demande d'interruption du 8279 à l'entrée série SID) :

	MVI	A, 00	
	STA	8279	ordre
\$o	RIM		
	RLC		
	JNC	\$o	
	MVI	A, 40	
	STA	8279	ordre
	LDA	8279	donnée : on lit la touche pressée

```

1          .TITLE  CARILL,'ON'
2
3          .MACRO  SIM
4          .BYTE  030
5          .ENDM
6
7          02E7  RDKBD  =002E7
8          205C  TON    =0205C
9          205B  LIMITE =0205B
10         205D  RYTHME =0205D
11         0060  LO     =060
12
13 0000      . =02000
14
15 2000 31C020  LXI     SP,020C0
16 2003 3E01    MVI     A,001
17 2005 D320    OUT     020
18 2007 32FF20  STA     020FF
19 200A 3E08    MVI     A,008
20 200C        SIM
21 200D FB      DEBUT:  EI
22 200E CDE702  CALL    RDKBD
23 2011 2E60    MVI     L,LO
24
                                $LO ADRESSE BASSE DU
                                $PREMIER MORCEAU

```


25	2013	07		RLC		;4 RLC
26	2014	07		RLC		;=MULTIPLICATION
27	2015	07		RLC		;PAR 16 SOIT 16 OCTETS
28	2016	07		RLC		;DONC 7 NOTES
29	2017	85		ADD	L	; (A)=L0+16*(A)
30	2018	6F		MOV	L,A	
31	2019	0610		ADI	010	; (A)=L0+16*(A)+10
32	201B	325B20		STA	LIMITE	
33	201E	7E		MOV	A,M	
34	201F	325C20		STA	TON	
35	2022	2C		INR	L	
36	2023	7E		MOV	A,M	
37	2024	325D20		STA	RYTHME	
38	2027	2C		INR	L	
39	2028	46	NOTE0:	MOV	B,M	; (B)=NOTE
40	2029	2C		INR	L	; (HL) POINTE LA DUREE
41	202A	3A5D20		LDA	RYTHME	
42	202D	57		MOV	D,A	
43	202E	4E	NOTE2:	MOV	C,M	; (C)=DUREE
44	202F	3EFF	NOTE1:	MVI	A,OFF	
45	2031	CD4F20		CALL	NOTE	
46	2034	CD4F20		CALL	NOTE	; (A) REVIENT NUL
47	2037	0D		DCR	C	
48	2038	C22F20		JNZ	NOTE1	
49	203B	15		DCR	D	
50	203C	C22E20		JNZ	NOTE2	
51	203F	06FF		MVI	B,OFF	;DETACHE LES NOTES
52	2041	CD5120		CALL	DET	
53	2044	2C		INR	L	
54	2045	3A5B20		LDA	LIMITE	
55	2048	BD		CMP	L	
56	2049	CA0D20		JZ	DEBUT	
57	204C	C32B20		JMP	NOTE0	
58						
59	204F	D321	NOTE:	OUT	021	
60	2051	3A5C20	DET:	LDA	TON	
61	2054	5F		MOV	E,A	
62	2055	78	NOTE4:	MOV	A,B	
63	2056	3D	NOTE3:	DCR	A	
64	2057	C25620		JNZ	NOTE3	
65	205A	1D		DCR	E	
66	205B	C25520		JNZ	NOTE4	
67	205E	C9		RET		

```

68
69
70 205F      .#02060
71
72           ;PREMIER MORCEAU
73
74 2060 02    .BYTE    002
75 2061 07    .BYTE    007
76 2062 9C    .BYTE    09C
77 2063 10    .BYTE    010
78 2064 75    .BYTE    075
79 2065 16    .BYTE    016
80 2066 75    .BYTE    075
81 2067 16    .BYTE    016
82 2068 5C    .BYTE    05C
83 2069 1B    .BYTE    01B
84 206A 5C    .BYTE    05C
85 206B 1B    .BYTE    01B
86 206C 75    .BYTE    075
87 206D 2B    .BYTE    02B
88 206E 9C    .BYTE    09C
89 206F 10    .BYTE    010
90
91           ;DEUXIEME MORCEAU
92
93 2070 04    .BYTE    004
94 2071 03    .BYTE    003
95 2072 8B    .BYTE    08B
96 2073 09    .BYTE    009
97 2074 8B    .BYTE    08B
98 2075 1B    .BYTE    01B
99 2076 8B    .BYTE    08B
100 2077 09    .BYTE    009
101 2078 68    .BYTE    068
102 2079 30    .BYTE    030
103 207A 68    .BYTE    068
104 207B 30    .BYTE    030
105 207C 5C    .BYTE    05C
106 207D 36    .BYTE    036
107 207E 5C    .BYTE    05C
108 207F 36    .BYTE    036

```

SERRURE ELECTRONIQUE

But : Utilisation des «Restarts».

Principales ou nouvelles instructions : JP, RZ

Ce programme concerne une serrure type «Digiclé». Il faut donner le code composé de quatre chiffres (1048320 combinaisons) parmi 16 possibles pour que la porte s'ouvre. Mais, dans notre cas, il faut donner rapidement la combinaison car on ne tolère que peu de temps de réflexion entre chaque chiffre, sinon on déclenche une sirène.

Dans ce programme, la porte fermée est symbolisée par l'affichage de CLAC, la porte ouverte par, et la sirène par le défilement de U. Cette sirène ne peut être interrompue que par VECT INTR (RST 7.5) qui referme également la porte en conservant le code. On utilise RST 5, 6 et 7 afin de gagner de l'espace mémoire.

Il faut noter que le programme de gestion de RST 7.5 nous envoie à une adresse dans le programme principal, ce qui nécessite un «recalage» du pointeur de pile. Ce programme permet la fermeture de la porte, le programme principal se terminant par HLT.

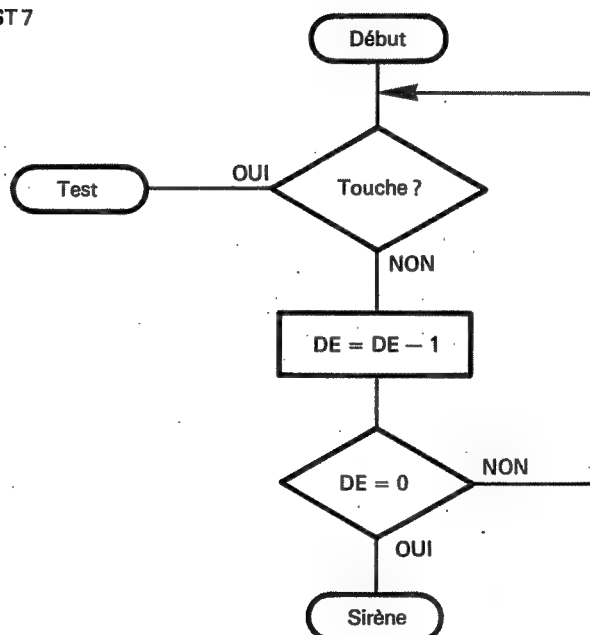
Le code est stocké en mémoire de 208B à 208E ; d'autre part, le sous-programme appelé par RST 7 ne renvoie au programme principal que si le chiffre est bon (RZ) ; sinon, il se prolonge par la sirène... Avant de lancer ce programme, ne pas oublier de charger les cases mémoires des RST n (restarts).

1			.TITLE	SERRUR, 'E ELECTRONIQUE'	
2					
3			.MACRO	SIM	
4			.BYTE	030	
5			.ENDM		
6					
7	02E7	RDKBD	=002E7		
8	05F1	DELAI	=005F1		
9	208B	RANG	=0208B		
10					
11	0000		..=02000		
12					
13	2000	31C020	LXI	SP, 020C0	
14	2003	3E08	MVI	A, 008	
15	2005		SIM		
16	2006	01BB20	LXI	B, RANG	
17	2009	F7	RST	6	#ENTREE
18	200A	F7	RST	6	
19	200B	F7	RST	6	
20	200C	F7	RST	6	#DU CODE
21	200D	2E30 DEBUT:	MVI	L, 030	#AFFICHAGE
22	200F	EF	RST	5	#DE "CLAC"
23	2010	FB	EI		
24	2011	CDE702	CALL	RDKBD	#PREMIERE TOUCHE
25	2014	2EBB	MVI	L, 08B	
26	2016	CD5B20	CALL	TEST	#COMPARAISON AVEC LE
27					#PREMIER CHIFFRE
28	2019	FF	RST	7	#DEUXIEME TOUCHE
29	201A	FF	RST	7	#LA TROISIEME
30	201B	FF	RST	7	#LA QUATRIEME
31	201C	2E36	MVI	L, 036	#AFFICHAGE
32	201E	EF	RST	5	#DE

33	201F	FB		EI		
34	2020	76		HLT		#RETOUR PAR V.I.
35						
36						
37	2021	110619	AFF#	LXI	D,01906	
38	2024	3E90		MVI	A,090	
39	2026	12		STAX	D	#ORDRE D'AFFICHAGE
40	2027	15		DCR	D	
41	2028	7E	#01	MOV	A,M	
42	2029	12		STAX	D	
43	202A	2C		INR	L	
44	202B	1D		DCR	E	
45	202C	C22B20		JNZ	#0	
46	202F	C9		RET		
47						
48	2030	6C		.BYTE	06C	"C"
49	2031	7C		.BYTE	07C	"L"
50	2032	8B		.BYTE	08B	"A"
51	2033	6C		.BYTE	06C	"C"
52	2034	FF		.BYTE	OFF	
53	2035	FF		.BYTE	OFF	
54	2036	F7		.BYTE	0F7	"."
55	2037	F7		.BYTE	0F7	
56	2038	F7		.BYTE	0F7	
57	2039	F7		.BYTE	0F7	
58	203A	F7		.BYTE	0F7	
59	203B	F7		.BYTE	0F7	
60						
61						
62	203C	FB	RGT:	EI		
63	203D	CDE702		CALL	RDKBD	
64	2040	02		STAX	B	#RANGEMENT DU CODE
65	2041	0C		INR	C	
66	2042	C9		RET		
67						
68			#ENTREE DES TOUCHES			
69						
70	2043	E5	ENT:	PUSH	H	#ON UTILISE HL DANS CE
71						#PROGRAMME
72	2044	FB		EI		
73	2045	2EFE		MVI	L,OFF	#CASE TAMPON DE RST5,5
74	2047	16FF		MVI	D,OFF	#TEMPORISATION
75	2049	7E	ENT1:	MOV	A,M	#TOUCHE DANS A
76	204A	B7		DRA	A	#POSITIONNE LES FLAGS
77	204B	F25720		JP	ENTO	#C'EST UNE TOUCHE
78	204E	1B		DCX	D	
79	204F	7B		MOV	A,E	
80	2050	B2		DRA	D	
81	2051	C24920		JNZ	ENT1	#LE DELAI N'EST PAS
82	2054	C35D20		JMP	SIRENE	#ECOULE, SINON
83	2057	36B0	ENTO:	MVI	M,0B0	#(CASE TAMPON) = 80H
84	2059	E1		POP	H	
85	205A	2C		INR	L	
86	205B	BE	TEST:	CMF	M	
87	205C	CB		RZ		#RETOUR SI BONNE CLEF
88	205D	3E09	SIRENE:	MVI	A,009	#N'AUTORISE QUE V.I.

89 205F		SIM		
90 2060 FB		EI		
91 2061 3E80	SIRE1:	MVI	A,080	
92 2063 4F		MOV	C,A	
93 2064 210619		LXI	H,01906	
94 2067 36CD	SIRE0:	MVI	M,0CD	#EXTINCTION DE CLAC
95 2069 1616		MVI	D,016	
96 206B CDF105		CALL	DELAI	
97 206E 71		MOV	M,C	
98 206F 25		DCR	H	
99 2070 361C		MVI	M,01C	#AFFICHE "U"
100 2072 1616		MVI	D,016	
101 2074 CDF105		CALL	DELAI	
102 2077 24		INR	H	
103 2078 0C		INR	C	
104 2079 2D		DCR	L	
105 207A C26720		JNZ	SIRE0	
106 207D C36120		JMP	SIRE1	
107				
108 2080 31C020	VECT:	LXI	SP,020C0	#ON RECALE SP
109 2083 210D20		LXI	H,DEBUT	#POUR LE RETOUR
110 2086 E5		PUSH	H	
111 2087 3E08		MVI	A,008	#AUTORISE LE CLAVIER
112 2089		SIM		
113 208A C9		RET		
114				
115 208B				
116				
117 20C2 C32120		JMP	AFF	#CASES DE RST5
118 20C5 C33C20		JMP	RGT	#CASES DE RST6
119 20C8 00		NOP		#CASES DE RST6.5
120 20C9 00		NOP		
121 20CA 00		NOP		
122 20CB C34320		JMP	ENT	#CASES DE RST7
123 20CE C38020		JMP	VECT	#CASES DE RST7.5

Organigramme de RST 7

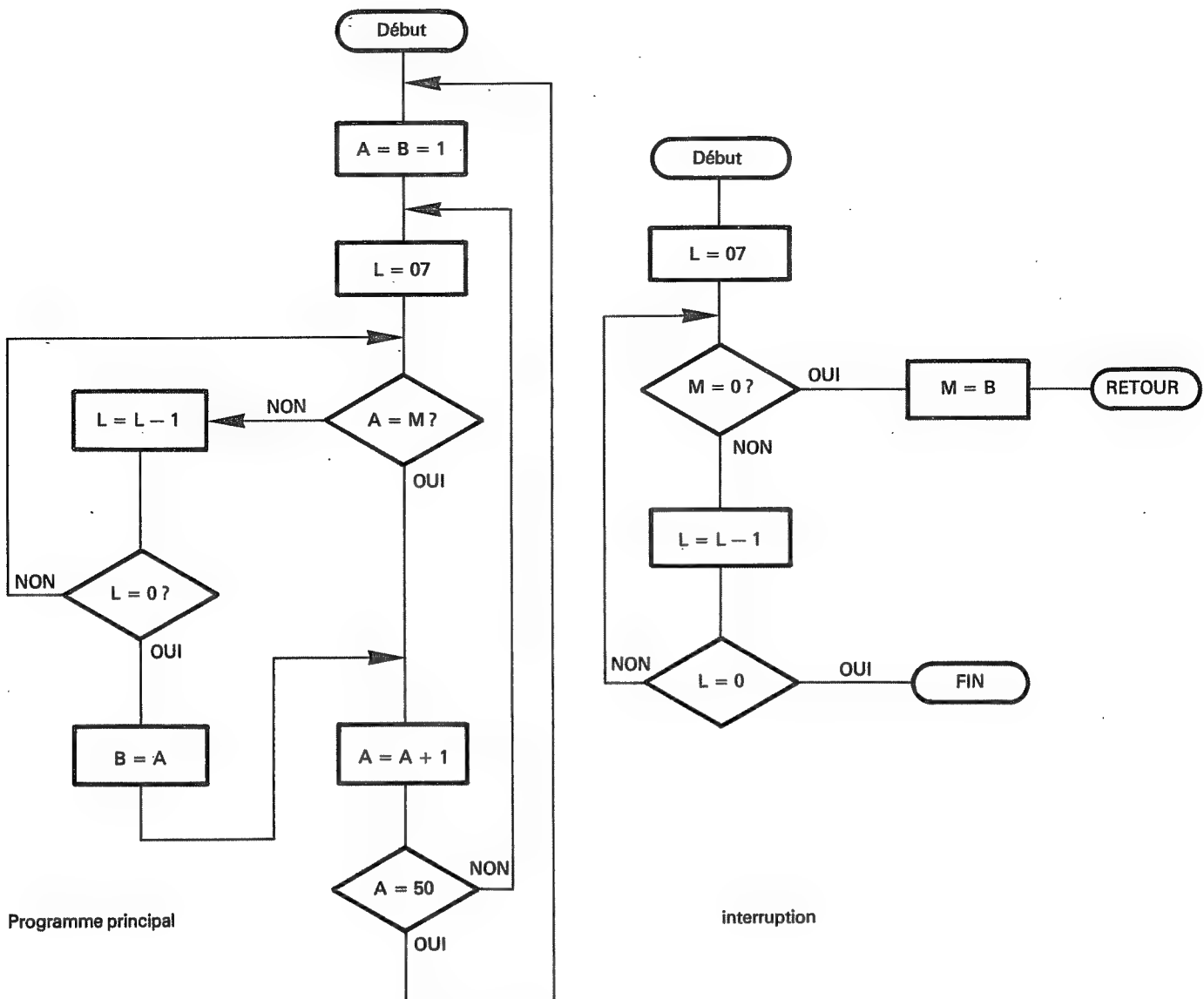


TIRAGE DU LOTO

Pour jouer au loto, il faut tirer 7 chiffres parmi 49. Nous effectuerons donc un comptage rapide de 1 à 49 avec boucle que nous interromprons 7 fois, le nombre contenu dans un registre au moment de l'interruption sera stocké en mémoire et *retiré* de la boucle de comptage. Le problème principal est de s'affranchir des rebonds de la touche *Vect-Intr* (on a expliqué pourquoi dans un programme précédent) ; ceci sera effectué à l'aide d'une temporisation de quelques centaines de millisecondes, avant de revenir au programme principal, et l'interruption ne sera autorisée qu'une fois par boucle.

Les organigrammes sont les suivants où :

- B est le registre-tampon ;
- M est une cellule de la mémoire de stockage adressée par HL, initialisée à 00.



L'interruption ne sera autorisée qu'un court instant après l'instruction MOV B,A qui garantit que le contenu de B n'a pas déjà été tiré.

Les cases-mémoires où sont stockés les nombres tirés sont placées de 2000 à 2007, de façon à simplifier le test sur le nombre de tirages. Le programme commence en 2010.

```

1      .TITLE  TIRAGE," DU LOTO"
2
3      .MACRO  SIM
4      .BYTE  030
5      .ENDM
6
7      036E    UPDDT    =0036E
8      05F1    DELAI    =005F1
9
10     0000     . =02010
11
12     2010 210720    LXI      H,02007
13     2013 AF        XRA      A                ; (A)=0
14     2014 77        MOV      M,A              ; INITIALISATION
15     2015 2D        DCR      L                ; DE 7 CASES
16     2016 C21420    JNZ      $0                ; MEMOIRES
17     2019 31C020    LXI      SP,020C0
18     201C 3E08      MVI      A,008
19     201E          SIM
20     201F 0601      BOUCLE: MVI      B,001
21     2021 78        MOV      A,B
22     2022 2E07      #2:      MVI      L,007
23     2024 BE        #1:      CMP      M                ; (A) A T'IL ETE TIRE ?
24     2025 CA2E20    JZ        $3                ; SI OUI
25     2028 2D        DCR      L
26     2029 C22420    JNZ      $1
27     202C 47        MOV      B,A                ; (A) N'A PAS ETE TIRE
28                                     ; ON LE RANGE DANS B
29     202D FB        EI                ; AUTORISATION DES
30                                     ; INTERRUPT., V.I EST
31                                     ; MEMORISEE
32     202E B7        #3:      ORA      A                ; CY=0
33     202F F3        DI                ; INTERRUPT. INTERDITES
34     2030 3C        INR      A
35     2031 27        DAA
36     2032 FE50      CPI      050                ; (A)=50 ?
37     2034 C22220    JNZ      $2
38     2037 C31F20    JMP      BOUCLE
39
40     203A 2E07      VECT:     MVI      L,007
41     203C AF        XRA      A                ; (A)=0
42     203D BE        #5:      CMP      M
43     203E CA4B20    JZ        $4                ; (CASE)=0 , CASE LIBRE
44     2041 2D        DCR      L
45     2042 C23D20    JNZ      $5
46     2045 C35320    JMP      AFF
47     2048 70        #4:      MOV      M,B                ; 7 CASES PLEINES
48     2049 1100F0    LXI      D,0F000            ; RANGE LE Nbre. TIRE
49     204C CDF105    CALL     DELAI            ; ANTI-REBOND POUR V.I.

```

50	204F	3E18	MVI	A,018	#R.A.Z DE U.I.
51	2051		SIM		
52	2052	C9	RET		
53	2053	2E07	MVI	L,007	
54	2055	E5	PUSH	H	#HL DETRUIT PAR UPDDT
55	2056	7E	MOV	A,M	
56	2057	CD6E03	CALL	UPDDT	
57	205A	0604	MVI	B,004	
58	205C	11FFFF	LXI	D,0FFFF	#TEMPORISATION
59	205F	CDF105	CALL	DELAI	
60	2062	05	DCR	B	
61	2063	C25C20	JNZ	#6	
62	2066	E1	POP	H	
63	2067	2D	DCR	L	
64	2068	C25520	JNZ	#7	
65	206B	76	HLT		
66					
67	206C			.=020CE	
68					
69	20CE	C33A20	JMP	VECT	
70					
71		0000		.END	

CREATION D'UN NOMBRE PSEUDO-ALEATOIRE

But : Générer un nombre dû au hasard, compris entre 0 et 100.

On se propose de créer un nombre aléatoire, plus exactement pseudo-aléatoire car la série obtenue, quoique longue, serait répétitive.

L'une des façons les plus simples pour obtenir une valeur aléatoire consiste exécuter un comptage rapide et à l'interrompre à un moment aléatoire, ainsi qu'on l'a fait dans le programme précédent. Une autre méthode revient à passer par une série d'opérations visant à créer le plus de possibilités. En voici un exemple, que l'on peut transformer à volonté.

On part d'une valeur aléatoire contenue par la cellule mémoire d'adresse 2040. On lui fait subir un traitement qui, tout d'abord élimine le chiffre 00, puis mène à une valeur définitive rangée en 2040 mais aussi affichée.

Il est certain qu'une telle méthode n'est que relativement satisfaisante ; son avantage principal réside dans le fait qu'elle n'occupe qu'une trentaine de cellules-mémoires pour le programme.

La paire H, L sert de pointeur. Les nombres obtenus couvrent de 1 à 99.

1	.TITLE ALEATO,'IRE'				
2					
3	#CREATION D'UN NOMBRE PSEUDO-ALEATOIRE				
4					
5	0000			.=02000	
6					
7	036E	UPDDT		=0036E	
8					
9	2000	31C020	LXI	SP,020C0	#INITIALISATION DE SP
10	2003	214020	LXI	H,02040	#CHARGEMENT POINTEUR
11	2006	0E00	MVI	C,000	
12	2008	7E	MOV	A,M	
13	2009	FE00	CPI	000	#POUR ELIMINER ZERO
14	200B	C21020	JNZ	SUITE	
15	200E	3EFF	MVI	A,OFF	#METTRE DES UNS
16	2010	47	SUITE: MOV	B,A	
17	2011	E61D	ANI	01D	#GARDONS QUELQUES BITS
18	2013	EA1820	JPE	PARITE	#AJOUTONS LA PARITE
19	2016	0EB0	MVI	C,0B0	
20	2018	78	PARITE: MOV	A,B	
21	2019	0F	RRC		
22	201A	E67F	ANI	07F	#BIT DE SIGNE = 0
23	201C	81	ADD	C	
24	201D	27	DAA		#AJUSTEMENT DECIMAL
25	201E	77	MOV	M,A	
26	201F	CD6E03	CALL	UPDDT	
27	2022	76	HLT		
28					
29	0000			.END	

HORLOGE 24 HEURES AVEC SONNERIE

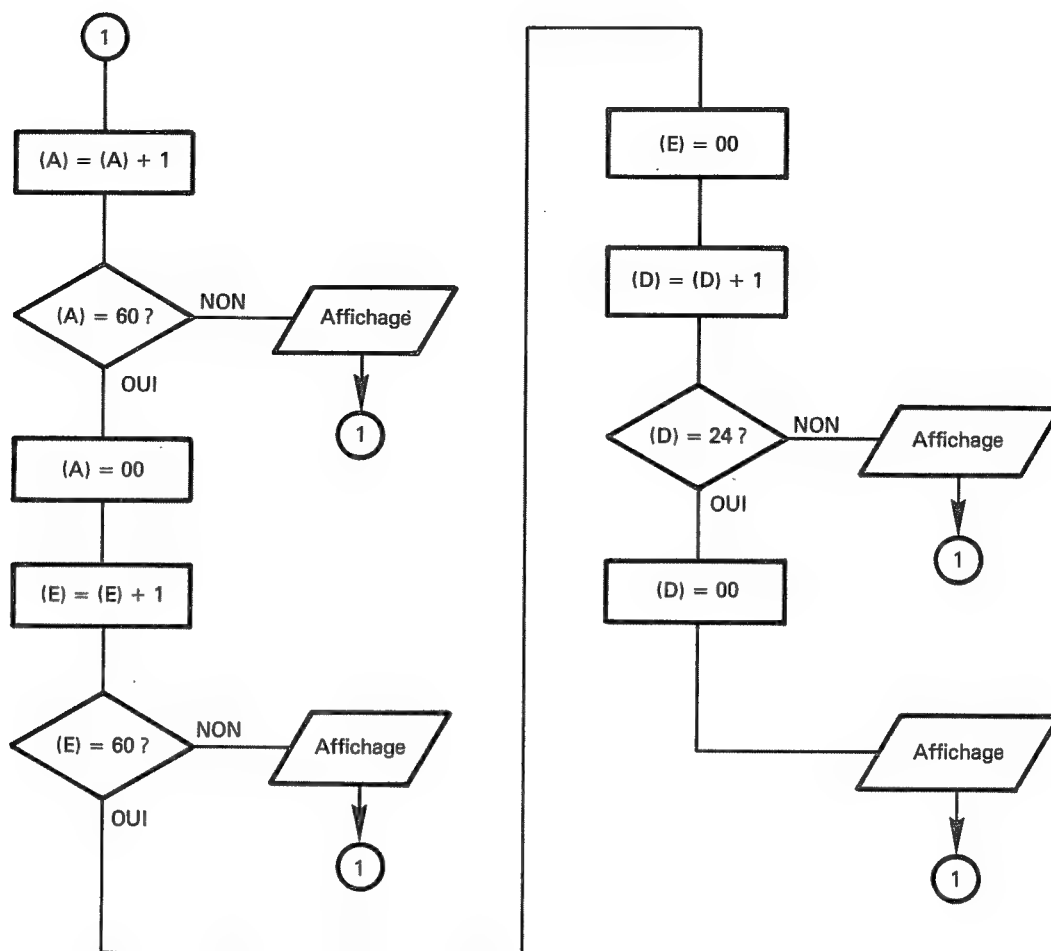
But : Organisation de multiples boucles participant à une séquence unique.

Principale ou nouvelle instruction utilisée : XTHL.

Ce programme affiche l'heure (minutes et secondes comprises) à partir du moment où on l'a initialisé. Il permet également la sonnerie à une heure donnée. Cette sonnerie est assurée par le timer de la deuxième RAM (supplémentaire) du kit, programmé pour émettre, jusqu'à interruption, un son. Le microprocesseur ne pouvant faire qu'une chose à la fois, nous lui donnons l'heure de la sonnerie avant l'heure exacte.

Dans l'organigramme de la boucle horaire :

- (A) représente les secondes,
- (E) les minutes et
- (D) les heures.



Le timer est commander en signaux carrés ; il faut donc que les deux bits de poids forts soient : 01. Le programme donne la quantité à utiliser pour avoir 1 kHz (0C00). La pile de sauvegarde et figurée ci-dessous :

20BA	Indicateurs	
20BB		A
20BC		E
20BD		D
20BE		L
20BF	H	
		secondes
		minutes
		heure
		minutes
		heure } sonnerie

```

1          .TITLE  HORLOG,"E"
2
3          #HORLOGE 24H,AVEC SONNERIE
4
5          .MACRO  SIM
6          .BYTE  030
7          .ENDM
8
9          02E7  RDKBD  =002E7
10         0363  UPDAD  =00363
11         036E  UPDDT  =0036E
12         05F1  DELAI  =005F1
13         F99C  BAT    =0CF99C          #A AJUSTER POUR AVOIR
14                                         #UNE HORLOGE "JUSTE"
15         0000  LSB    =000          #TIMBRE DE SONNERIE
16         004C  MSB    =04C          #A REGLER (IL FAUT
17                                         #MSB=01XX XXXX )
18
19
20 0000          . =02000
21
22 2000 31C020    LXI      SP,020C0
23 2003 3E08      MVI      A,008
24 2005          SIM                      #INTERRUPTIONS
25 2006 FB        EI                      #AUTORISEES
26 2007 FF        RST      7              #ENTREE HEURE SONNERIE
27 2008 67        MOV      H,A          #
28 2009 E5        PUSH     H              # (H) DETRUIT PAR RST7
29 200A FF        RST      7              #MINUTES SONNERIE
30 200B E1        POP      H
31 200C 6F        MOV      L,A
32 200D E5        PUSH     H
33 200E FF        RST      7
34 200F 57        MOV      D,A          #ENTREE HEURE EXACTE ,
35 2010 D5        PUSH     D
36 2011 FF        RST      7
37 2012 D1        POP      D
38 2013 5F        MOV      E,A          #MINUTES EXACTES ,
39 2014 D5        PUSH     D
40 2015 FF        RST      7              #SECONDES EXACTES .

```

41	2016	D1	DEBUT:	POF	D	
42	2017	E1		POF	H	
43	2018	47		MOV	B,A	
44	2019	7B		MOV	A,E	
45	201A	AD		XRA	L	#MIN.=MIN-SONNERIE ?
46	201B	C22620		JNZ	#0	
47	201E	7A		MOV	A,D	
48	201F	AC		XRA	H	#HEURE=HEURE-SONNERIE ?
49	2020	C22620		JNZ	#0	
50	2023	CD6820		CALL	SONRIE	#OUI
51	2026	7B	#0:	MOV	A,B	
52	2027	E5		PUSH	H	
53	2028	D5		PUSH	D	
54	2029	F5		PUSH	PSW	
55	202A	CD6303		CALL	UPDAD	#AFFICHE HEURE,MINUTES
56	202D	F1		POF	PSW	
57	202E	F5		PUSH	PSW	
58	202F	CD6E03		CALL	UPDDT	#AFFICHE SECONDES
59	2032	0602		MVI	B,002	#BOUCLE
60	2034	119CF9	BOUCLE:	LXI	D,BAT	#
61	2037	CDF105		CALL	DELAI	#DE
62	203A	05		DCR	B	#
63	203B	C23420		JNZ	BOUCLE	#1 SECONDE
64	203E	CD4520		CALL	TEMPS	
65	2041	F1		POF	PSW	
66	2042	C31620		JMP	DEBUT	
67						
68	2045	21BA20	TEMPS:	LXI	H,020BA	#(HL) POINTE LA CASE
69						#PRECEDANT CELLE QUI
70						#CONTIENT LES SECONDES
71	2048	1E60		MVI	E,060	
72	204A	CD5620		CALL	CALCUL	
73	204D	CD5620		CALL	CALCUL	
74	2050	1E24		MVI	E,024	
75	2052	CD5620		CALL	CALCUL	
76	2055	C9	FIN:	RET		
77						
78	2056	2C	CALCUL:	INR	L	
79	2057	7E		MOV	A,M	#(A)=SECONDES,PUIS
80						#MINUTES,PUIS HEURE ,
81						#SUIVANT L'INSTANT DE
82						#L'APPEL DE "CALCUL"
83	2058	3C		INR	A	#N'AFFECTE PAS CY
84	2059	B7		ORA	A	#CY=0
85	205A	27		DAA		
86	205B	BB		CMP	E	#(A)=60 OU 24 ?
87	205C	C26220		JNZ	RETOUR	
88	205F	AF		XRA	A	#SI (A)=60 OU 24 ALORS
89	2060	77		MOV	M,A	#SECONDES,MINUTES OU
90						#HEURE = 0

91	2061	C9		RET		
92	2062	77	RETOUR:	MOV	M+A	#ON REMET EN PLACE LA
93						#GRANDEUR INCREMENTEE
94						#ET ON TERMINE "TEMPS"
95						#EN CHARGEANT LA FILE
96	2063	215520		LXI	H+FIN	#AVEC L'ADRESSE DE
97	2066	E3		XTHL		#FIN DE "TEMPS"
98	2067	C9		RET		
99						
100	2068	FB	SONRIE:	EI		#POUR ARRET
101	2069	3E00		MVI	A+LSB	
102	206B	D32C		OUT	02C	
103	206D	3E4C		MVI	A+MSB	
104	206F	D32D		OUT	02D	
105	2071	3EC0		MVI	A+0C0	#DEPART SONNERIE
106	2073	D32B		OUT	02B	
107	2075	C9		RET		
108						
109	2076	3E40	ARRET:	MVI	A+040	#ARRET IMMEDIAT
110	207B	D32B		OUT	02B	#DU TIMER
111	207A	C9		RET		
112						
113	207B	CDE702	ENTREE:	CALL	RDKBD	
114	207E	F5		PUSH	PSW	
115	207F	CD6E03		CALL	UPDDT	
116	2082	F1		POP	PSW	
117	2083	07		RLC		
118	2084	07		RLC		
119	2085	07		RLC		
120	2086	07		RLC		
121	2087	47		MOV	B+A	
122	208B	FB		EI		
123	2089	CDE702		CALL	RDKBD	
124	208C	80		ADD	B	
125	208D	F5		PUSH	PSW	
126	208E	CD6E03		CALL	UPDDT	
127	2091	F1		POP	PSW	
128	2092	FB		EI		
129	2093	C9		RET		
130						
131	2094			..=020CB		
132						
133	20CB	C37B20		JMP	ENTREE	#ADRESSE DE RST7
134	20CE	C37620		JMP	ARRET	#ADRESSE DE RST7.5
135						
136		0000		.END		

BATAILLE NAVALE

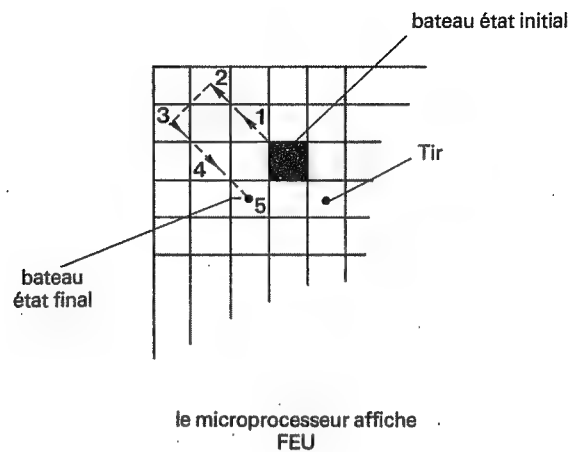
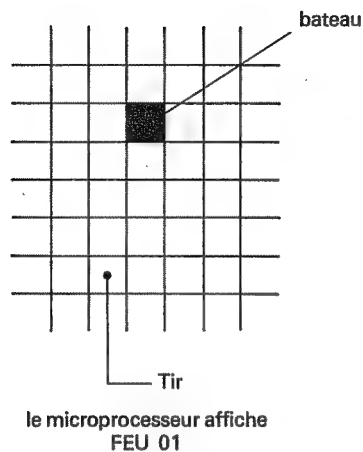
But : Accroître la longueur des programmes avec des possibilités multiples.

Ce programme de « bataille navale » consiste en la recherche des coordonnées d'un bateau occupant une case sur un échiquier de 256 (16×16) cases. Pour augmenter les difficultés, le bateau « fuit » si le tir à lieu dans l'une des 8 cases entourant sa position. La fuite à lieu dans la direction opposée au tir et le bateau se déplace de 5 cases avec rebondissement sur les parois. Un joueur donne la position du bateau après que le microprocesseur ait affiché JOUE. Le deuxième joueur peut tirer dès que FEU est affiché. Le microprocesseur indique en « data » la distance minimale de tir (en x ou y), n'indique rien si le bateau fuit, et affiche COULE si le bateau est touché. On revient au début par l'ordre « VECT INTR ». Les schémas donnent deux cas de figures.

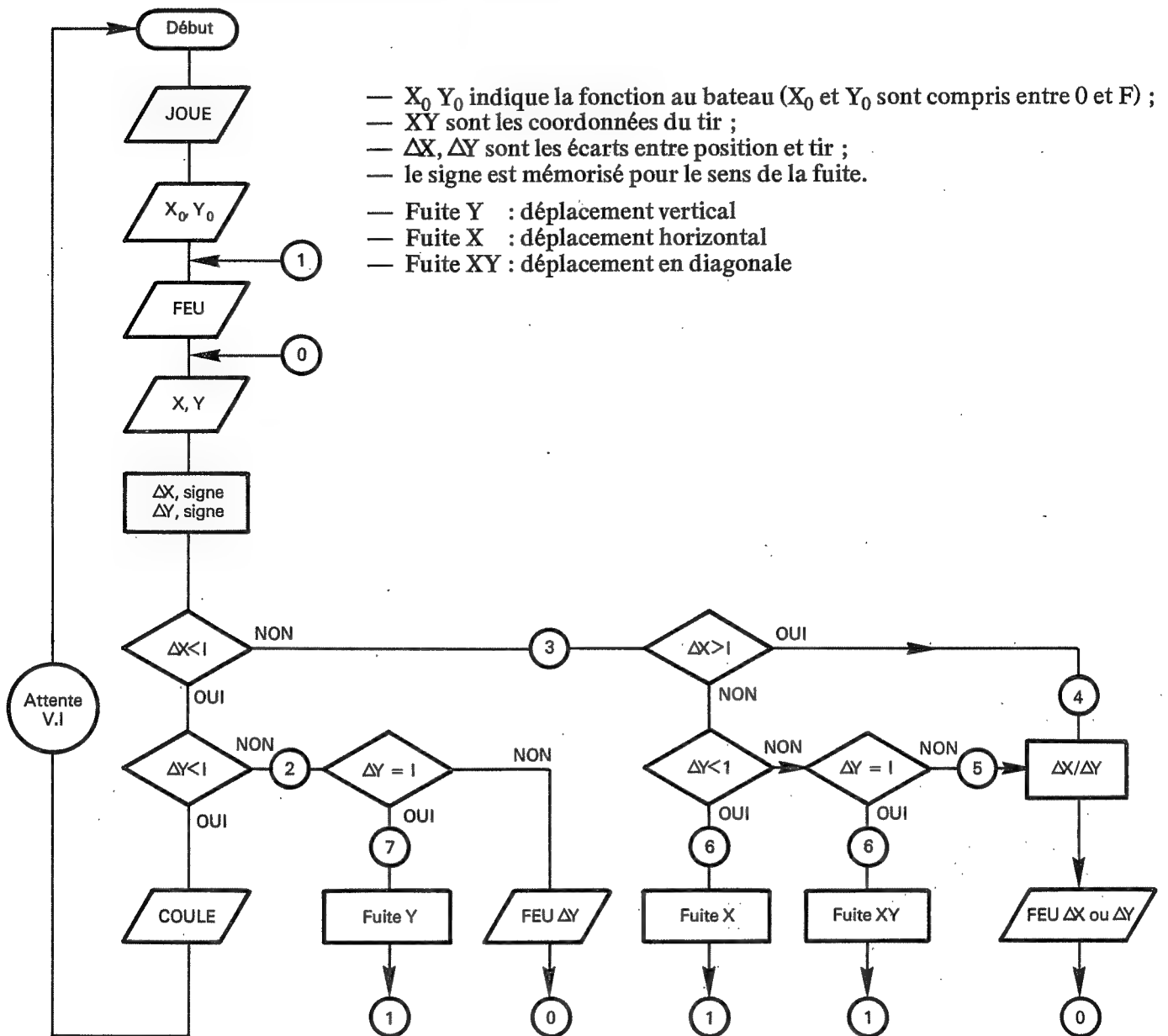
1			„TITLE	BATAIL„LE NAVALE„	
2					
3			„MACRO	SIM	
4			„BYTE	030	
5			„ENDM		
6					
7	02E7	RDKBD	=002E7		
8	036E	UPDDT	=0036E		
9					
10	0000		„=02000		
11					
12	2000	31C220 DEBUT:	LXI	SP,020C2	
13	2003	219420	LXI	H„JOUE	„ADRESSE DE „JOUE„
14	2006	EF	RST	5	„AFFICHE „JOUE„
15	2007	3E08	MVI	A„008	
16	2009		SIM		
17	200A	FB	EI		
18	200B	F7	RST	6	„APPEL DE TOUCHE
19	200C	47	MOV	B„A	„B CONTIENT X„
20	200D	FB	EI		
21	200E	F7	RST	6	
22	200F	4F	MOV	C„A	„C CONTIENT Y„
23	2010	219A20 „1:	LXI	H„FEU	„ADRESSE DE „FEU„
24	2013	EF	RST	5	
25	2014	FB „0:	EI		
26	2015	F7	RST	6	
27	2016	57	MOV	D„A	„D CONTIENT X
28	2017	FB	EI		
29	2018	F7	RST	6	„A CONTIENT Y
30	2019	91	SUB	C	„(A) = ECART EN Y „DY
31	201A	F22120	JP	DYP	„DY>0
32	201B	2F	CMA		
33	201E	3C	INR	A	

34	201F	2E01		MVI	L,001	#DY<0 (L)=1
35	2021	5F	DYP:	MOV	E,A	#(E)= DY
36	2022	7A		MOV	A,D	
37	2023	90		SUB	B	#(A)=ECART EN X :DX
38	2024	F22B20		JP	DXF	
39	2027	2F		CMA		
40	2028	3C		INR	A	
41	2029	2601		MVI	H,01	#DX<0 (H)=1
42	202B	57	DXP:	MOV	D,A	#(A)=(D)= DX
43	202C	FE01		CPI	001	# DX =1 : CY=0
44	202E	D24220		JNC	\$3	
45	2031	7B		MOV	A,E	
46	2032	FE01		CPI	001	#TEST DY
47	2034	D23B20		JNC	\$2	
48	2037	21A020		LXI	H,COULE	#ADRESSE DE COULE
49	203A	EF		RST	5	
50	203B	FB		EI		
51	203C	76		HLT		#ATTENTE U.I.
52	203D	CA6620	\$2:	JZ	\$7	# DY =1 FUITE EN Y
53	2040	AF		XRA	A	
54	2041	FF		RST	7	# DY >1 ET DX =0 ,
55						#AFFICHAGE DE 00
56	2042	C25220	\$3:	JNZ	\$4	
57	2045	7B		MOV	A,E	# DX =1 DY =?
58	2046	FE01		CPI	001	# DY <1 CY=1
59	204B	DA5C20		JC	\$6	# DY =0 FUITE EN X
60	204B	C25120		JNZ	\$5	# DY >1 ET DX =1 ,
61						#AFFICHAGE DE 01
62	204E	C35C20		JMP	\$6	# DX = DY =1 FUITE
63						# EN X ET Y
64	2051	7A	\$5:	MOV	A,D	
65	2052	BB	\$4:	CMP	E	
66	2053	CA5B20		JZ	\$8	# DX = DY AFFICHE
67	2056	D25A20		JNC	\$9	# DX > DY
68	2059	FF		RST	7	#AFFICHE DX
69	205A	7B	\$9:	MOV	A,E	
70	205B	FF	\$8:	RST	7	#AFFICHE DY
71	205C	7B	\$6:	MOV	A,B	
72	205D	CD7120		CALL	FUITE	#FUITE EN X
73	2060	47		MOV	B,A	
74	2061	7B		MOV	A,E	
75	2062	B7		ORA	A	
76	2063	CA6C20		JZ	\$10	#DY=0
77	2066	79	\$7:	MOV	A,C	
78	2067	65		MOV	H,L	
79	206B	CD7120		CALL	FUITE	#FUITE EN Y
80	206B	4F		MOV	C,A	
81	206C	2620	\$10:	MVI	H,020	#POUR LE RETOUR
82	206E	C31020		JMP	\$1	
83						
84	2071	25	FUITE:	DCR	H	
85	2072	C27F20		JNZ	\$11	#DX>0
86	2075	C605		ADI	005	#X ₀ =X ₀ +5
87	2077	FE10		CPI	010	#X ₀ +5>10 ?
88	2079	D8		RC		#SI CY=1 RETOUR
89	207A	C6E2		ADI	0E2	

90	207C	2F		CMA		
91	207D	3C		INR	A	#CONVERSION SI CY=0
92	207E	C9		RET		
93	207F	C4FB	#11:	ADI	OFB	#X ₀ =X ₀ -5
94	2081	F0		RF		#RETOUR SI X ₀ -5>0
95	2082	2F		CMA		
96	2083	3C		INR	A	
97	2084	C9		RET		
98						
99	2085	110619	AFF:	LXI	D,01906	
100	2088	3E90		MVI	A,090	
101	208A	12		STAX	D	
102	208B	15		DCR	D	
103	208C	7E	#12:	MOV	A,M	
104	208D	12		STAX	D	
105	208E	2C		INR	L	
106	208F	1D		DCR	E	
107	2090	C28C20		JNZ	#12	
108	2093	C9		RET		
109						
110	2094	1E	JOUE:	.BYTE	01E	
111	2095	0C		.BYTE	00C	
112	2096	1C		.BYTE	01C	
113	2097	6B		.BYTE	06B	
114	2098	FF		.BYTE	0FF	
115	2099	FF		.BYTE	0FF	
116	209A	EB	FEU:	.BYTE	0EB	
117	209B	6B		.BYTE	06B	
118	209C	1C		.BYTE	01C	
119	209D	FF		.BYTE	0FF	
120	209E	FF		.BYTE	0FF	
121	209F	FF		.BYTE	0FF	
122	20A0	6C	COULE:	.BYTE	06C	
123	20A1	0C		.BYTE	00C	
124	20A2	1C		.BYTE	01C	
125	20A3	7C		.BYTE	07C	
126	20A4	6B		.BYTE	06B	
127	20A5	FF		.BYTE	0FF	
128						
129	20A6	C5	ECART:	PUSH	B	#SAUVE (BC)=X ₀ ,Y ₀
130	20A7	CD6E03		CALL	UPDDT	
131	20AA	C1		FDP	B	
132	20AB	211420		LXI	H,#0	#ADRESSE DE RETOUR
133	20AE	E3		XTHL		
134	20AF	C9		RET		
135						
136	20B0			.=020C2		
137						
138	20C2	C3B520		JMP	AFF	#RST5 AFFICHAGE MOT
139	20C5	C3E702		JMP	RDKBD	#RST6 APPEL DE TOUCHE
140	20C8	00		NOP		#EMPLACEMENT
141	20C9	00		NOP		#DE
142	20CA	00		NOP		#RST 6.5
143	20CB	C3A620		JMP	ECART	#RST7
144	20CE	C30020		JMP	DEBUT	#V.I.
145						
146		0000		.END		



Pour gagner de la place en mémoire, on utilise les RST X. L'organigramme (très important, ici, pour gagner des octets, est le suivant :



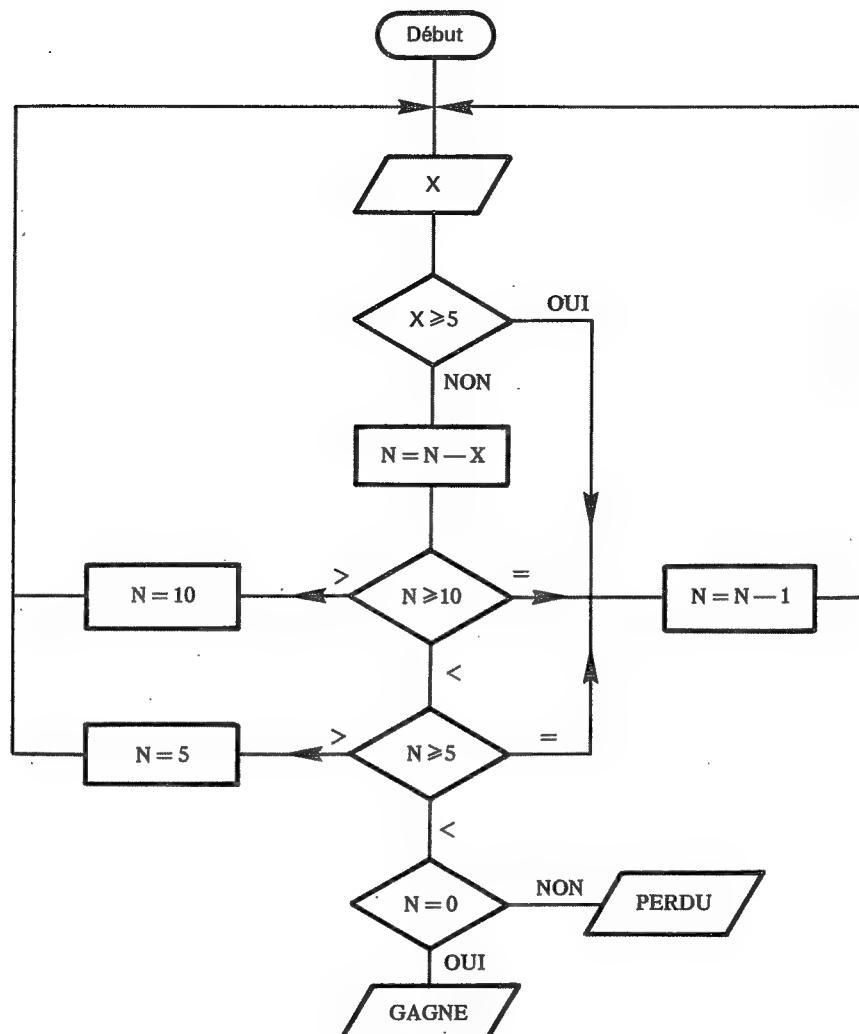
JEU DE NIM

But : Résoudre un délicat problème d'affichage.

Ce jeu d'allumettes consiste à partir de 16 allumettes ; chaque joueur en retire au maximum 4 et celui qui retire la dernière a perdu.

La méthode, *pour gagner*, consiste à *laisser à l'adversaire 6 allumettes*, d'où l'organigramme suivant pour la machine qui, très intelligente, opère un raccourci dès qu'il reste moins de 5 allumettes.

Au début, la machine joue si l'on presse une touche de valeur supérieure ou égale à 5 (X est la touche pressée) ; le nombre d'allumettes restant est $N + 1$



```

1          .TITLE  NIM
2
3          ;JEU DE NIM ,ON ENLEVE AU PLUS 4 ALLUMETTES DU TAS
4          ;IL NE FAUT PAS PRENDRE LA DERNIERE.
5
6          .MACRO  SIM
7          .BYTE  '030'
8          .ENDM
9
10         02E7  RDKBD  =002E7
11         05F1  DELAI  =005F1
12         FFFF  TEMPO  =0FFFF
13
14         0000          . =02000
15
16         2000 31C020          LXI      SP,020C0
17         2003 019720          LXI      B,02097          ;CASE MEMOIRE POUR N
18         2006 3E0B          MVI      A,00B
19         200B          SIM
20         2009 3E0F          MVI      A,00F          ;N=15
21         200B FF          RST      7          ;APPEL RANG.,AFFICH.,
22                                     ;TEMPORISATION
23         200C FB          DEBUT: EI
24         200D CDE702          CALL    RDKBD          ;ATTENTE DE TOUCHE
25         2010 6F          MOV      L,A          ;CODE X SAUVE DANS L
26         2011 FE05          CPI      005          ;SI X)=5 ,LA MACHINE
27         2013 D23720          JNC      MACH          ;COMMENCE
28         2016 0A          LDAX     B          ;N RAPPELE DANS A
29         2017 95          SUB      L          ;N=N-X
30         2018 FF          RST      7
31         2019 0A          LDAX     B
32         201A FE0A          CPI      00A          ;0A=10D ,SI N=10 ,
33         201C CA3720          JZ       MACH          ;LA MACHINE RETIRE
34                                     ;1 ALLUMETTE
35         201F DA2520          JC       SUITE          ;N<10 ,SI N>10 ,LA
36         2022 3E0A          MVI      A,00A          ;MACHINE FAIT N=10
37         2024 F7          RST      6          ;APPEL RANG.,AFFICH.,
38                                     ;TEMPO. ET SAUT EN
39                                     ;DEBUT
40         2025 FE05          SUITE: CPI      005          ;N)=5 ?
41         2027 CA3720          JZ       MACH          ;N=5
42         202A DA3020          JC       FIN          ;N<5 ,SI N>5 ,LA
43         202D 3E05          MVI      A,005          ;MACHINE FAIT N=5
44         202F F7          RST      6
45         2030 B7          FIN:   ORA      A          ;N<5 ON POSITIONNE
46                                     ;LES FLAGS
47         2031 C27520          JNZ      PERDU          ;N#0
48         2034 C3B620          JMP      GAGNE          ;N=0
49         2037 0A          MACH:  LDAX     B
50         203B 3D          DCR      A          ;N=N-1
51         2039 F7          RST      6
52
53         203A 02          AFF:   STAX     B          ;ON SAUVE N
54         203B 210B19          LXI      H,0190B          ;POUR AFFICHAGE

```

55	203E	11FFFF	TEMP:	LXI	D,TEMPO	
56	2041	CDF105		CALL	DELAI	
57	2044	2D		DCR	L	
58	2045	C23E20		JNZ	TEMP	#SI TEMP TROP GRAND
59						#DIMINUER L
60	2048	36CD		MVI	M,0CD	#EXTINCTION
61	204A	11FF14		LXI	D,014FF	
62	204D	CDF105		CALL	DELAI	#ATTENTE POUR AFFICHER
63	2050	3690		MVI	M,090	
64	2052	25		DCR	H	
65	2053	367F		MVI	M,07F	#"
66	2055	0A		LDAX	B	
67	2056	6F	BOUCLE:	MOV	L,A	#SAUVE N DANS L
68	2057	FE03		CPI	003	#N)=3 ?
69	2059	D26D20		JNC	N3	#N)=3
70	205C	FE01		CPI	001	#N)=1 ?
71	205E	CA6A20		JZ	N1	#N=1
72	2061	DA6720		JC	NO	#N(1
73	2064	367B		MVI	M,07B	#"" N=2
74	2066	C9		RET		
75	2067	36FF	NO:	MVI	M,0FF	#"" N=0
76	2069	C9		RET		
77	206A	367F	N1:	MVI	M,07F	#"" N=1
78	206C	C9		RET		
79	206D	366B	N3:	MVI	M,06B	#"" N=3
80	206F	7D		MOV	A,L	#N REMIS DANS A
81	2070	D603		SUI	003	#N=N-3
82	2072	C35620		JMP	BOUCLE	
83						
84	2075	24	PERDU:	INR	H	#(H) VALAIT 18
85	2076	3690		MVI	M,090	
86	2078	25		DCR	H	
87	2079	36CB		MVI	M,0CB	#"P"
88	207B	366B		MVI	M,06B	#"E"
89	207D	36FA		MVI	M,0FA	#"r"
90	207F	361A		MVI	M,01A	#"d"
91	2081	361C		MVI	M,01C	#"u"
92	2083	36FF		MVI	M,0FF	#""
93	2085	76		HLT		
94						
95	2086	24	GAGNE:	INR	H	
96	2087	3690		MVI	M,090	
97	2089	25		DCR	H	
98	208A	362C		MVI	M,02C	#"G"
99	208C	36BB		MVI	M,0BB	#"A"
100	208E	362C		MVI	M,02C	#"G"
101	2090	36BA		MVI	M,0BA	#"n"
102	2092	366B		MVI	M,06B	#"E"
103	2094	36FF		MVI	M,0FF	#""
104	2096	76		HLT		
105						
106	2097					
107						

108	20C5	210C20	LXI	H, DEBUT	†RST6
109	20C8	E3	XTHL		†RST6.5
110	20C9	7F	MOV	A, A	†EQUIVALENT
111	20CA	7F	MOV	A, A	†A NOP NOP
112	20CB	C33A20	JMP	AFF	†RST7
113					
114		0000	.END		

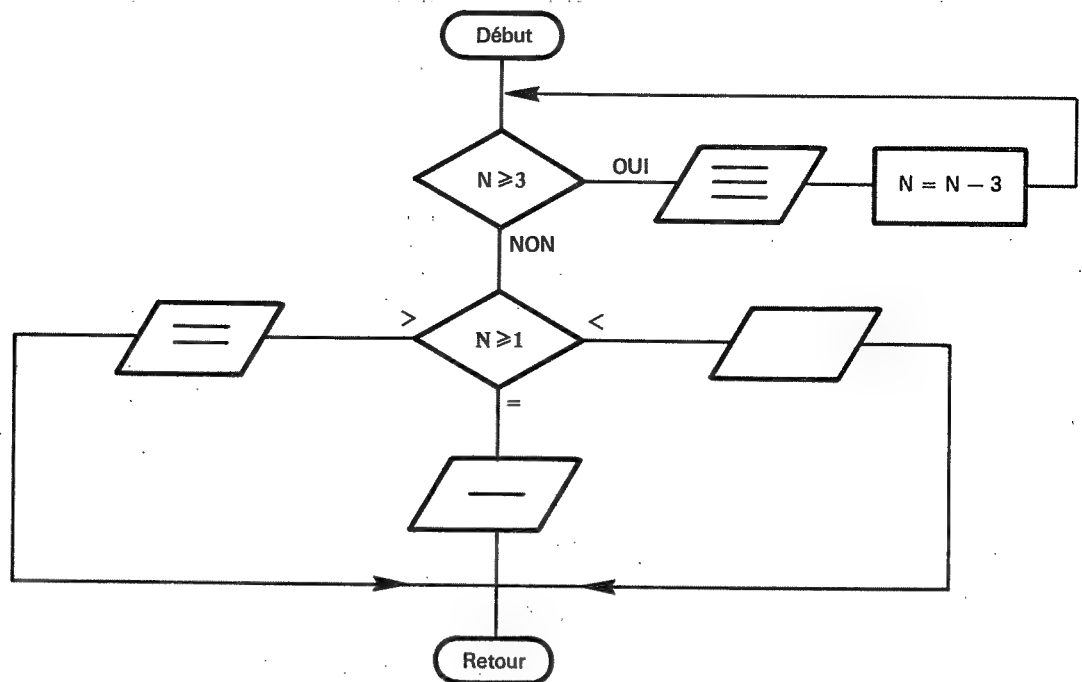
Pour ce programme, la difficulté réside dans l'affichage. Les allumettes vont être figurées par des segments horizontaux. Nous aurons donc au début 5 paquets de trois allumettes plus une :

```

      — — — — —
— — — — —
      — — — — —

```

qui disparaîtront petit à petit. L'organigramme d'affichage est donc (N est le nombre d'allumettes) :

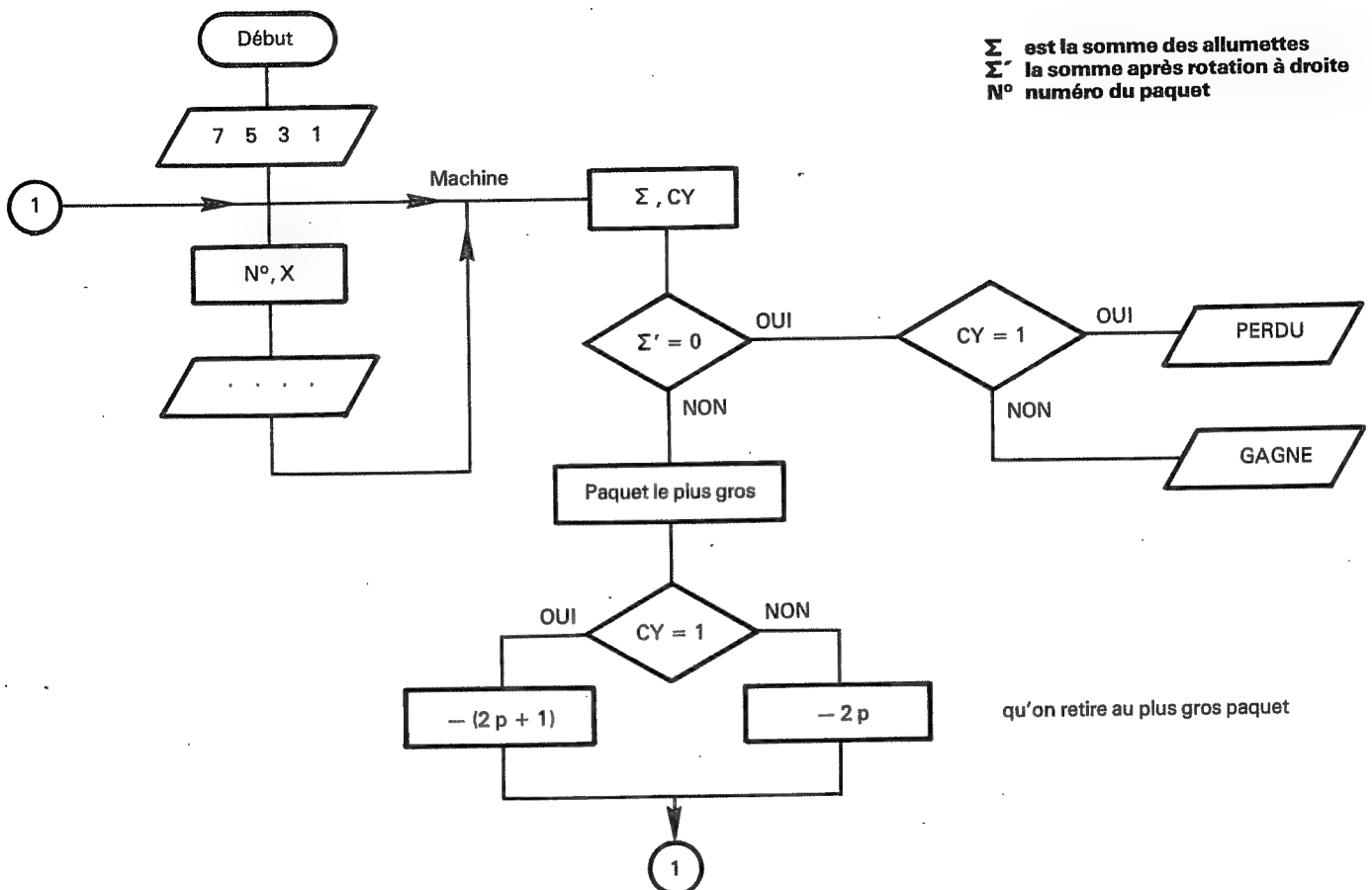


Les boucles sont temporisées : il faut attendre quelques secondes avant l'apparition (ou la disparition) des allumettes, et laisser à la machine le temps de «réfléchir». On peut accélérer le jeu en modifiant le contenu de l'adresse 203C

JEU DE MARIENBAD

But : Un autre jeu, pour souffler....

C'est un autre jeu d'allumettes qui compte également 16 objets, mais répartis en quatre paquets de 7, 5, 3, 1 allumettes. Dans cette variante du jeu célèbre de Marienbad, il faut retirer dans *un* paquet de 1 à 7 allumettes. Pour tenter de gagner, il faut laisser à l'adversaire un nombre *impair* d'allumettes (*ne pas penser à utiliser JPO et JPE, car elles ne concernent que la parité du nombre de 1*) ; le perdant est celui qui retire la dernière allumette. Le test de la parité est fait à l'aide d'une rotation à droite avec test sur le «carry» (indicateur de retenue). L'organigramme est le suivant :



```

1          .TITLE  MARIEN,"BAD"
2
3          $JEU DE MARIENBAD ,ON ENLEVE AUTANT D'ALLUMETTES
4          $QUE L'ON VEUT D'UN PAQUET ,IL NE FAUT PAS PRENDRE
5          $LA DERNIERE
6
7          .MACRO  SIM
8          .BYTE  030
9          .ENDM
10
11      02E7  RDKBD    =002E7
12      02B7  OUTPT    =002B7
13      034E  UPDDT    =0034E
14      05F1  DELAI    =005F1
15
16      0000          . =02000
17
18      2000 31C520 DEBUT: LXI      SP,020C5          $REMARQUEZ C5 - ON
19                                     $N'UTILISE PAS RST5
20      2003 AF          XRA      A
21      2004 CD6E03      CALL     UPDDT          $AFFICHE "00"
22      2007 210705      LXI      H,00507
23      200A 22B020      SHLD     020B0          $ (20B0)=7, (20B1)=5
24      200D 210301      LXI      H,00103
25      2010 22B220      SHLD     020B2          $ (20B2)=3, (20B3)=1
26      2013 3E1B      MVI      A,01B          $AUTORISATION DES
27      2015          SIM          $INTERRUPTIONS ,AVEC
28      2016 FB      EI          $R.A.Z. DE V.I.
29      2017 FF      RST      7          $APPEL RDKBD (TOUCHE)
30      201B B7      ORA      A          $POSITIONNE LES FLAGS
31      2019 C22E20      JNZ      $0          $ 0 ON COMMENCE
32      201C CDB220 $9:  CALL     AFF
33      201F FB      EI
34      2020 FF      RST      7
35      2021 4F      MOV      C,A          $N DU PAQUET DANS C
36      2022 FB      EI
37      2023 FF      RST      7          $NOMBRE A RETIRER
38      2024 0600      MVI      B,000
39      2026 21AF20      LXI      H,020AF
40      2029 09      DAD      B          $ (HL)=ADRES. DU PAQUET
41      202A 2F      CMA
42      202B 3C      INR      A
43      202C B6      ADD      M          $ON RETIRE LE NOMBRE
44                                     $D'ALLUMETTES VOULU
45      202D 77      MOV      M,A          $LE RESTE EST RANGE
46      202E CDB220 $0:  CALL     AFF
47      2031 F7      RST      6          $TEMPORISATION
48      2032 21B020      LXI      H,020B0          $HL DETRUIT PAR OUTPT
49      2035 7E      MOV      A,M
50      2036 23      INX      H          $OU INR L
51      2037 B6      ADD      M          $
52      203B 23      INX      H          $ON FAIT LA SOMME DES
53      2039 B6      ADD      M          $
54      203A 23      INX      H          $

```

55	203B	B6	ADD	M	#PAQUETS : S
56	203C	1F	RAR		#CY=1 S EST IMPAIRE
57					#CY=0 S EST PAIRE
58	203D	47	MOV	B,A	#SALVE S,ROT
59	203E	3E00	MVI	A,000	#ON GARDE CY
60	2040	17	RAL		
61	2041	4F	MOV	C,A	#CY DANS C (C)=0 OU 1
62	2042	7B	MOV	A,B	
63	2043	B7	DRA	A	
64	2044	CABB20	JZ	FIN	#IL RESTAIT MOINS DE
65					# 2 ALLUMETES
66	2047	2EB0	MVI	L,0B0	
67	2049	5D	MOV	E,L	#SALVE (L)
68	204A	0603	MVI	B,003	
69	204C	7E	MOV	A,M	#RECHERCHE DU PLUS
70	204D	23	INX	H	#
71	204E	BE	CMP	M	#
72	204F	D25420	JNC	#2	#
73	2052	5D	MOV	E,L	#GROS
74	2053	7E	MOV	A,M	#
75	2054	05	DCR	B	#
76	2055	C24D20	JNZ	#3	#PAQUET
77	205B	6B	MOV	L,E	#(HL)=ADRESSE DU PLUS
78					#GROS PAQUET , (A)=NBRE.
79					#D'ALLUMETTES
80	2059	0D	DCR	C	
81	205A	C27320	JNZ	SPAIRE	#(C)=0 S EST PAIRE
82	205D	FE05	CPI	005	#(A)>=5 ? TRAITEMENT
83	205F	DA6720	JC	#5	#
84	2062	C6FC	ADI	0FC	#(A)=(A)-4
85	2064	C37E20	JMP	#6	#
86	2067	C6FE	ADI	0FE	#(A)=(A)-2 POUR
87	2069	FA6F20	JM	#7	#(A)-2(0 ?
88	206C	C37E20	JMP	#6	#
89	206F	AF	XRA	A	#(A)=0
90	2070	C37E20	JMP	#6	# S IMPAIRE
91	2073	FE04	CPI	004	#(A)>=4 ? TRAITEMENT
92	2075	DA7D20	JC	#B	#
93	207B	C6FD	ADI	0FD	#(A)=(A)-3 POUR
94	207A	C37E20	JMP	#6	#
95	207D	3D	DCR	A	#(A)=(A)-1 S PAIRE
96	207E	77	MOV	M,A	
97	207F	C31C20	JMP	#9	
98					
99	2082	21B020	LXI	H,020B0	
100	2085	AF	XRA	A	# 4 SIGNES
101	2086	47	MOV	B,A	#PAS DE POINT
102	2087	CDB702	CALL	OUTPT	
103	208A	C9	RET		
104					
105	208B	2619	MVI	H,019	
106	208D	3690	MVI	M,090	
107	208F	25	DCR	H	
108	2090	0D	DCR	C	

109	2091	CAA220	JZ	\$10	#(C) VALAIT 1.
110	2094	36CB	MVI	M, 0CB	#"P"
111	2096	36DB	MVI	M, 0DB	#"E"
112	2098	36FA	MVI	M, 0FA	#"F"
113	209A	361A	MVI	M, 01A	#"d"
114	209C	363E	MVI	M, 03E	#"H"
115	209E	36FF	MVI	M, 0FF	
116	20A0	FB	EI		
117	20A1	76	HLT		#POUR REPRISE AVEC V.I.
118	20A2	362C	MVI	M, 02C	#"G"
119	20A4	36DB	MVI	M, 0DB	#"A"
120	20A6	362C	MVI	M, 02C	#"G"
121	20A8	36DA	MVI	M, 0DA	#"n"
122	20AA	36DB	MVI	M, 0DB	#"E"
123	20AC	36FF	MVI	M, 0FF	
124	20AE	FB	EI		
125	20AF	76	HLT		
126					
127	20B0			.=020B4	
128					
129	20B4	24	INTR:	INR	H
130	20B5	36CD	MVI	M, 0CD	#EXTINCTION
131	20B7	F7	RST	6	#TEMPORISATION
132	20B8	C30020	JMP	DEBUT	
133					
134	20BB			.=020C5	
135					
136	20C5	16FF	MVI	D, 0FF	#RST6
137	20C7	CDF105	CALL	DELAI	#PLACE DE RST6.5 NON
138	20CA	C9	RET		#UTILISEE
139	20CB	C3E702	JMP	RDKBD	#RST7 EST UN CALL
140	20CE	C3B420	JMP	INTR	#V.I.
141					
142		0000		.END	

Au départ, le microprocesseur affiche 00 en donnée ; le jeu commence dès que l'on a pressé une touche avec : 0, on commence, sinon c'est la machine qui démarre. Pour afficher les combinaisons, on utilise le sous-programme moniteur OUTPT qui commence à 02B7 (ce programme est utilisé par UPDAD et UPDDT) ; il faut stocker les signes à afficher dans des cases mémoires contiguës, initialiser HL à l'adresse du premier caractère à afficher, mettre (A) à 0 si l'on affiche en «adresse», à 1 pour les «données» ; si (B) vaut 0, il n'y aura pas de point mais par contre, si (B) vaut 1, il y aura un point à droite du dernier signe affiché. Pour afficher «perdu» ou «gagné», dont les caractères ne sont pas «traduits» par le moniteur, on écrit un programme qui envoie les codes des signes à afficher au 8279.

MASTER MIND

Il s'agit de trouver une combinaison de 4 chiffres compris entre 0 et 9 par essais successifs. La machine répond par le numéro de l'essai et le nombre de chiffres «bons et bien placés» (B/BP) et «bons mais mal placés» (B/MP). Le «choix» de la combinaison est effectué par tirage (voir programme du loto).

Ce tirage peut donner des doubles, par exemples :

1, 1, 8, 0

si on essaie «1, 2, 8,0», la machine répond :

XE 30 : Xème essai 3 : B/BP, 0 : B/MP

Par contre, à l'essai «2, 9, 1, 4», la machine répond :

YE 02 : Yème essai 0 : B/BP, 2 : B/MP

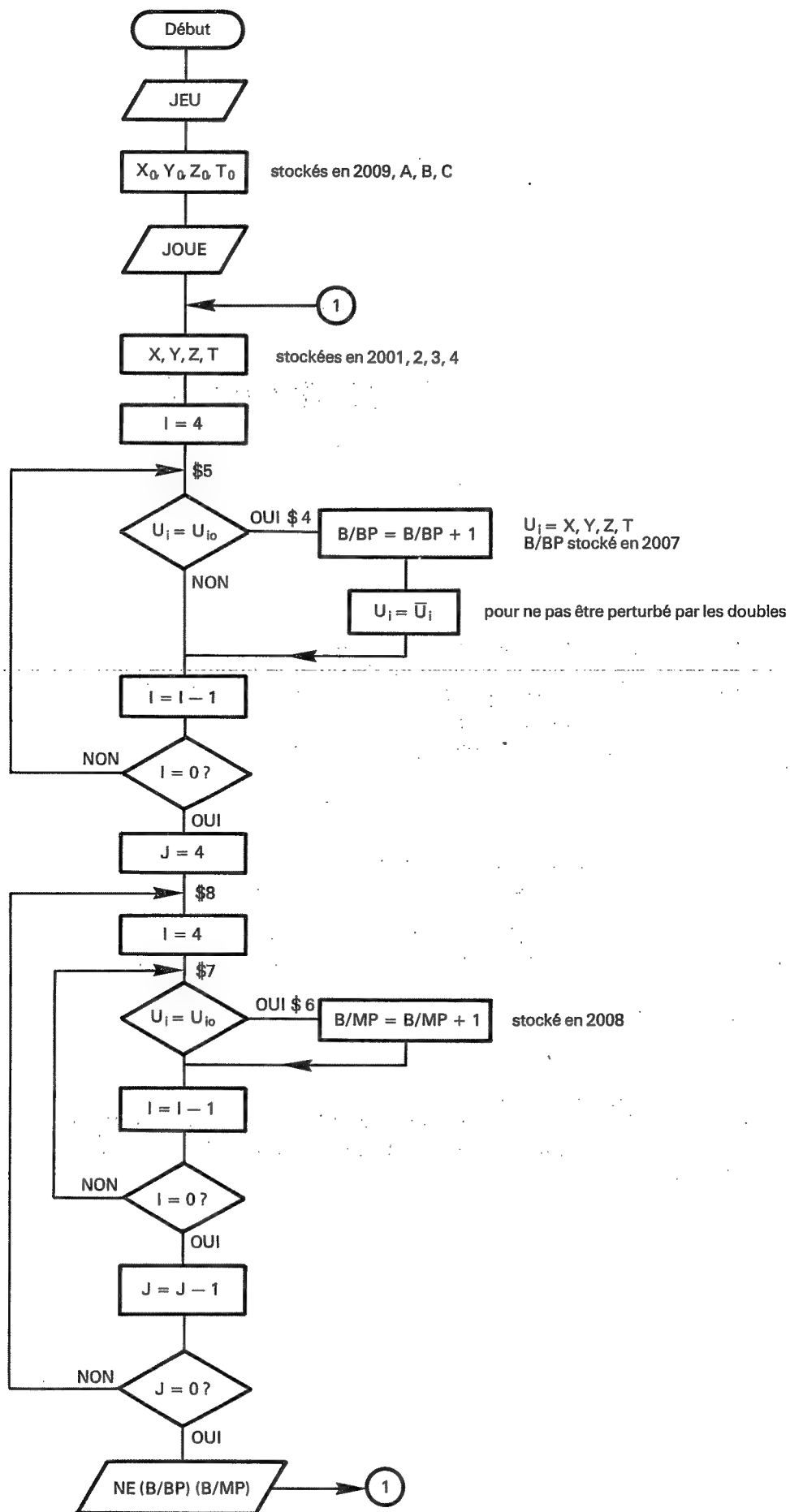
L'essai est bon si la machine répond :

UE 40

On reprend le jeu par «Reset».

Comme pour le programme précédent on utilise les procédés d'affichage.

Attention : il y a un «raccourci», le programme principal est la suite du programme d'interruption !



1			.TITLE MASTER,"-MIND"		
2					
3			.MACROD SIM		
4			.BYTE 030		
5			.ENDM		
6					
7	02E7	RDKBD	=002E7		
8	02B7	OUTPT	=002B7		
9	05F1	DELAI	=005F1		
10	FFFF	TEMPO	=0FFFF		
11					
12	0000		.=0200D		
13					
14	200D	31C020	LXI SP,020C0		
15	2010	01A220	LXI B,020A2		#ADRES. DU J DE "JEU"
16	2013	FF	RST 7		#AFFICHE "JEU" ON PEUT
17	2014	0E04	MVI C,004		#TIRER 4 CHIFFRES
18	2016	210920	LXI H,02009		#ADRESSE DE X0
19	2019	F3 #1:	DI		#EVITE XI=18
20	201A	3E18	MVI A,018		#MASQ.INTR. AVEC R.A.Z
21	201C		SIM		#DE V.I.
22	201D	3E09	MVI A,009		
23	201F	FB	EI		
24	2020	3D #0:	DCR A		#(A)=(A)-1
25	2021	C22020	JNZ #0		
26	2024	C31920	JMP #1		
27					
28	2027	77 JEU:	MOV M,A		#ON ARRIVE ICI PAR V.I.
29	2028	F5	PUSH PSW		#(A) RANGE ET SAUVE
30	2029	11FFFF	LXI D,TEMPO		#ANTI-REBONDS
31	202C	CDF105	CALL DELAI		
32	202F	F1	POP PSW		
33	2030	2C	INR L		#ON INCREMENTE (L) POUR
34					#LE TIR SUIVANT
35	2031	0D	DCR C		
36	2032	C0	RNZ		#RETOUR SI 4 CHIFFRES
37					#N'ONT PAS ETE TIRES
38	2033	2E05	MVI L,005		
39	2035	71	MOV M,C		#(2005)=NBRE ESSAIS=0
40	2036	2C	INR L		
41	2037	360E	MVI M,00E		#(2006)=E POUR AFFICH.
42	2039	2C	INR L		
43	203A	71	MOV M,C		#B/BP=00 EN 2007
44	203B	2C	INR L		
45	203C	71	MOV M,C		#B/MP=00 EN 2008
46	203D	0EA6	MVI C,0A6		#"JOUER" : ON PEUT
47	203F	FF	RST 7		#ESSAYER 4 CHIFFRES
48	2040	010420 DEBUT:	LXI B,02004		#OUTPT DETRUIT (BC)
49	2043	51	MOV D,C		
50	2044	3E15	MVI A,015		#ON RANGE 15 EN
51	2046	02 #2:	STAX B		#2001,2002,2003,2004
52	2047	0D	DCR C		#POUR NE RIEN
53	2048	C24620	JNZ #2		#AFFICHER
54	204B	3E0C	MVI A,00C		#INTERDIT V.I.

55 204D		SIM		
56 204E FB	\$3:	EI		
57 204F CDE702		CALL	RDKBD	#MET 20FE DANS HL
58 2052 0C		INR	C	
59 2053 02		STAX	B	#X RANGE EN 2001,2,3,4
60 2054 C5		PUSH	B	#OUTPT DETRUIT (BC)
61 2055 D5		PUSH	D	#ET (DE)
62 2056 2E01		MVI	L,001	#POUR AFFICHAGE
63 2058 F7		RST	6	#DE L'ESSAI
64 2059 D1		POP	D	
65 205A C1		POP	B	
66 205B 15		DCR	D	
67 205C C24E20		JNZ	\$3	#ICI (BC)=2004
68 205F 11FFFF		LXI	D,TEMPO	#TEMPORISATION
69 2062 CDF105		CALL	DELAI	#(D)=(E)=0
70 2065 210C20		LXI	H,0200C	#ADRESSE DE TO
71 2068 0A	\$5:	LDAX	B	#T DANS A
72 2069 BE		CMP	M	#T=TO ?
73 206A C27020		JNZ	\$4	
74 206D 14		INR	D	#T=TO DONC B/BP=B/BP+1
75 206E 2F		CMA		#COMPLEMENTE T POUR NE
76				#PAS LE COMPTER B/MF
77 206F 02		STAX	B	
78 2070 2D	\$4:	DCR	L	
79 2071 0D		DCR	C	
80 2072 C26820		JNZ	\$5	#TEST NON TERMINER
81 2075 2D		DCR	L	#(HL)=2007
82 2076 72		MOV	M,D	#NBRE DE B/BP RANGE
83 2077 1604		MVI	D,004	#J=4
84 2079 2E0C		MVI	L,00C	
85 207B 0E04	\$8:	MVI	C,004	#I=4
86 207D 0A	\$7:	LDAX	B	
87 207E BE		CMP	M	
88 207F C28320		JNZ	\$6	
89 2082 1C		INR	E	#B/MF=B/MF+1
90 2083 0D	\$6:	DCR	C	
91 2084 C27D20		JNZ	\$7	
92 2087 2D		DCR	L	
93 2088 15		DCR	D	
94 2089 C27B20		JNZ	\$8	
95 208C 73		MOV	M,E	#NBRE DE B/MF RANGE
96 208D 2E05		MVI	L,005	
97 208F 34		INR	M	#NBRE ESSAIS : N
98				#INCREMENTE
99 2090 F7		RST	6	#AFFICHE N E B/BP B/MF
100 2091 C34020		JMP	DEBUT	
101				
102 2094 210419	AFF:	LXI	H,01904	#4 LETTRES A AFFICHER
103 2097 3690		MVI	M,090	
104 2099 25		DCR	H	
105 209A 0A	\$9:	LDAX	B	
106 209B 77		MOV	M,A	
107 209C 0C		INR	C	
108 209D 2D		DCR	L	

```

109 209E C29A20      JNZ      #9
110 20A1 C9          RET
111
112 20A2 1E          .BYTE    01E      #'J'
113 20A3 6B          .BYTE    06B      #'E'
114 20A4 1C          .BYTE    01C      #'U'
115 20A5 FF          .BYTE    0FF
116 20A6 1E          .BYTE    01E      #'J'
117 20A7 0C          .BYTE    00C      #'O'
118 20A8 1C          .BYTE    01C      #'U'
119 20A9 6B          .BYTE    06B      #'E'
120
121 20AA              . = 020C5
122
123 20C5 AF          XRA      A          #RST6 (A)=0 : 4 LETTRES
124 20C6 47          MOV      B,A        #PAS DE POINT
125 20C7 CDB702      CALL     OUTPT
126 20CA C9          RET
127                  #ON UTILISE LA PLACE RESERVEE A RST6.5
128
129 20CB C39420      JMP      AFF        #RST7
130 20CE C32720      JMP      JEU        #V.I.
131
132                0000      .END

```

ALGORITHME DE TRI

But - Décrire la méthode de base servant à classer des grandeurs dans l'ordre.

Ce type de programme est très classique : il s'agit de classer les nombres contenus dans un tableau dans l'ordre, croissant ou décroissant, au choix. Pour cela, il faut successivement examiner toutes les valeurs du tableau et exécuter les permutations nécessaires.

Le tableau lui-même, de 9 nombres dans notre exemple, est stocké en mémoire de 2041 à 2049 ; l'adresse 2040, elle, contient le nombre d'éléments du tableau, soit 9. Supposons qu'on veuille effectuer le classement dans l'ordre décroissant : après exécution de ce programme, on retrouvera toutes ces valeurs aux mêmes adresses, mais classées.

Le processus est le suivant. On examine les nombres par paires, d'abord les deux premiers ; s'ils sont dans l'ordre, on n'y touche pas ; sinon, on les intervertit. L'indication, «ordre» ou «désordre» est notée dans un registre témoin, ici (B) avec 0 s'il n'y a pas eu de permutation, et 1 s'il y en a eu une. On va voir que le rangement complet demande plusieurs passes. L'adresse en mémoire est stockée dans la paire H, L.

Prenons un exemple : soit à ranger dans l'ordre décroissant les 4 nombres 20, 30, 10, 40 rangés en 2041 à 2044. On met d'abord (B) à zéro et on charge (H, L) avec 2041. Le premier nombre, 20, est appelé dans l'accumulateur et (H,L) est incrémenté de 1. Puis, on compare (A) du contenu de l'adresse fournie par (H, L), donc 2042 où l'on trouve 30 : il faut donc permuter ces deux valeurs, ce qu'indique l'indicateur de retenue qui s'est mis à 1.

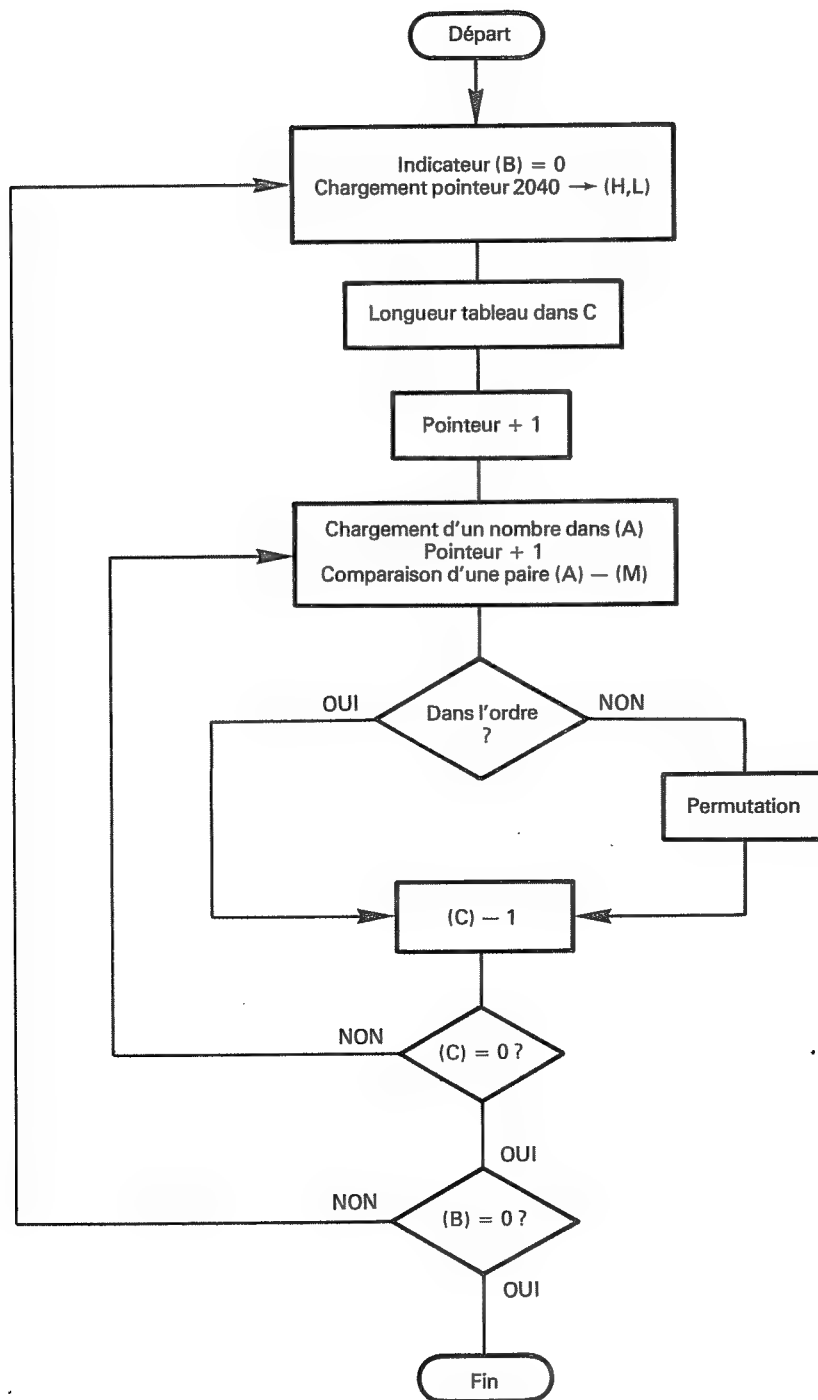
Pour cela, on transfère (2042) dans le registre D qui contiendra donc 30, et on profite de ce que (H, L) pointent toujours 2042 pour y envoyer (A), donc 20. On décrémente (H, L) et on expédie à 2041 le contenu du registre D : le tour est joué. On incrémente (H, L) et on stocke 1 dans le témoin B.

Puis, on passe à la paire suivante, donc 20 en 2042, et 10 en 2043 : là, tout va bien. On examine alors la dernière paire, 10 et 40 qui exige une permutation. Mais à son issue et à la fin de la première passe, on s'aperçoit que le classement n'est pas encore le bon puisqu'on a obtenu 30, 20, 40, 10. La valeur 40 doit encore remonter de deux niveaux, et il faudra encore deux passes : c'est précisément le registre indicateur B qui, chaque fois qu'il contient 1 à la fin d'une passe, sert à en commander une autre. Pour l'utiliser, on soustrait 1 de (B) ; si le résultat est nul, c'est qu'il contenait bien 1 ; lorsque le résultat ne sera plus nul (il sera alors négatif, avec 0-1, mais cela n'a aucun intérêt), cela signifiera que le tri est terminé.

Il est intéressant de constater, d'autre part, qu'on peut réduire de 1 le total des nombres examinés à chaque passe puisque le dernier sera «chassé» dès la première, etc. Le nombre total des passes est donc au maximum de N-1, ou N est le total des nombres.

On se méfiera d'un piège : l'égalité de deux nombres, que doit alors faire le programme de tri ? S'il les permute, ce pourrait être sans fin... C'est bien pourquoi on a exploité ici l'instruction de branchement conditionnel JNC, branchement s'il n'y a pas de retenue après la comparaison, donc si (M) est *supérieur* ou *égal* à (A) pour spécifier qu'il n'y a pas de permutation.

L'organigramme et son programme sont développés sur ces principes. Si, à titre d'exercice, vous avez rangé en mémoire (2040) = 09 pour indiquer qu'il y a 9 nombres, et dans les 9 cellules suivantes, les valeurs 3C, 7F, A9, 21, 83, 54, 06, C8 et 99, après exécution du programme vous trouverez C8, A9, 99, 83, 7F, 54, 3C, 21 et 06.



LOC	OBJ	LINE	SOURCE STATEMENT
		1	NAME CLASSEMENT
		2	
		3	;CLASSEMENT DE DONNEES PAR ORDRE DECROISSANT
		4	
2000		5	ORG 2000H
		6	
2000 0600		7	TRI: MVI B,0H ;"FLAG" DE PERMUTATION
2002 214020		8	LXI H,2040H ;POINTEUR
2005 4E		9	MOV C,M ;COMBIEN DE NOMBRES ?
2006 0D		10	DCR C ;... DE COMPARAISONS ?
2007 23		11	INX H ;DEBUT DU TABLEAU
2008 7E		12	TOURS: MOV A,M ;APPEL D'UN NOMBRE...
2009 23		13	INX H
200A BE		14	CMP M ;COMPARE AU SUIVANT
200B D21520		15	JNC BON ;DANS L'ORDRE ?
200E 56		16	MOV D,M ;DANS LE DESORDRE
200F 77		17	MOV M,A ;PERMUTER !
2010 2B		18	DCX H
2011 72		19	MOV M,D
2012 23		20	INX H
2013 0601		21	MVI B,01H ;POSITIONNER LE FLAG
2015 0D		22	BON: DCR C ;PREPARER TEST SUIVANT
2016 C20820		23	JNZ TOURS
2019 05		24	DCR B
201A C20020		25	JNZ TRI ;OU LA PASSE SUIVANTE
201D CF		26	RST 1
		27	
2000		28	END TRI

CALCUL ET INTRODUCTION DE LA PARITE

But - Traiter, par logiciel, la «parité».

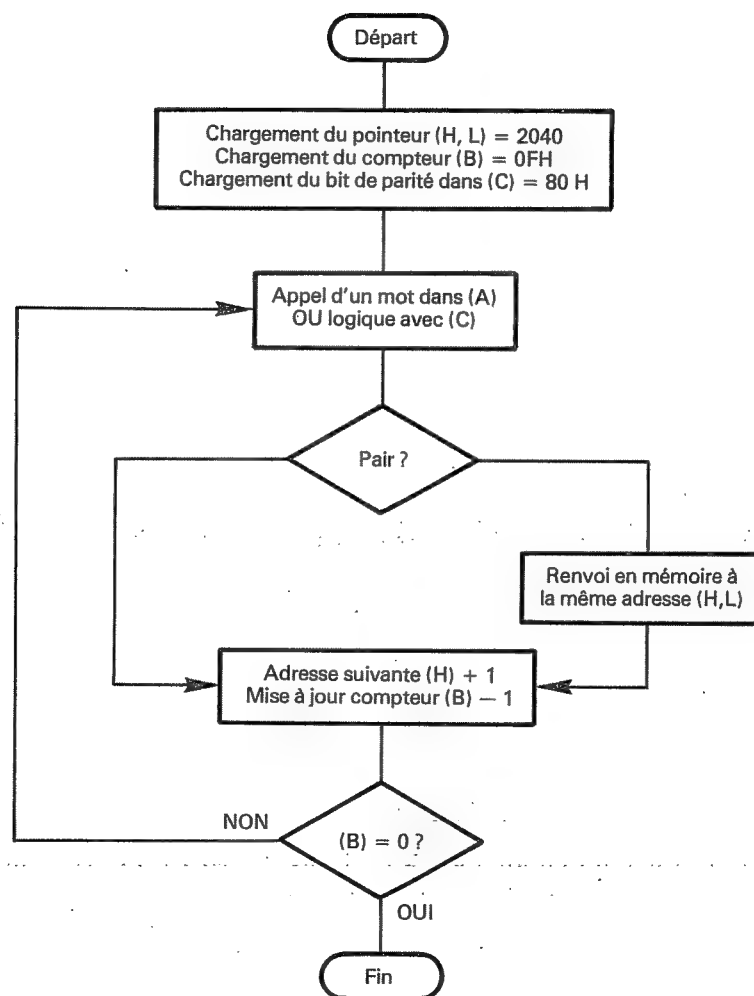
Le bit de parité que l'on ajoute à un mot de 7 bits, tel que celui qu'émet ou reçoit un télé-imprimeur, sert à vérifier que la transmission n'a pas été entachée d'erreur. Si l'on émet un mot *pair*, il faut que le mot reçu soit pair, lui aussi, *la parité étant comprise ici de la façon suivante : le total des 1 du mot est pair.*

Comment procède-t-on alors pour introduire un bit de parité dans un mot du code ASCII (sur 7 bits) ? Soit une chaîne de mots rangés en mémoire, de 2040 à 204 E, par exemple, qui seront émis vers un téléimprimeur ; en code ASCII, ils sont sur 7 bits ce qui signifie que leur bit de plus fort poids est obligatoirement un zéro.

L'adresse de départ du tableau est chargée dans la paire (H, L), sa longueur (en nombre de cellules) dans le registre B, donc 0F en hexadécimal. On commence par charger le bit de parité dans un registre, C par exemple, qui reçoit donc 80 (hexa), équivalent du binaire 1000 0000.

Puis, le premier mot est appelé dans l'accumulateur et on lui ajoute d'office le bit de parité à l'aide d'un OU logique entre (A) et (C), avec retour du résultat dans A. Suit un branchement conditionnel, puisque le 8085 possède un indicateur de parité : si le mot résultant est impair, on l'abandonne et on passe à la suite ; en effet, c'est que le mot d'origine était déjà pair. Si, par contre, on a créé la parité, on le ré-expédie en mémoire et à la même adresse. Puis on passe au suivant, et ce jusqu'à la fin de la chaîne.

Pour vérifier ce programme, introduisez en mémoire à partir de 2040 les codes 41, 42, 43, 44, etc. jusqu'à 4F qui correspondent aux codes ASCII des lettres A à O, et lancez l'exécution. Vous trouverez ensuite en mémoire 41, 42, C3, 44, C5, C6, 47, 48, C9, CA, 4B, CC, 4D, 4E, CF, ce qui peut vous surprendre mais une prompte vérification vous prouvera que c'est rigoureusement correct.

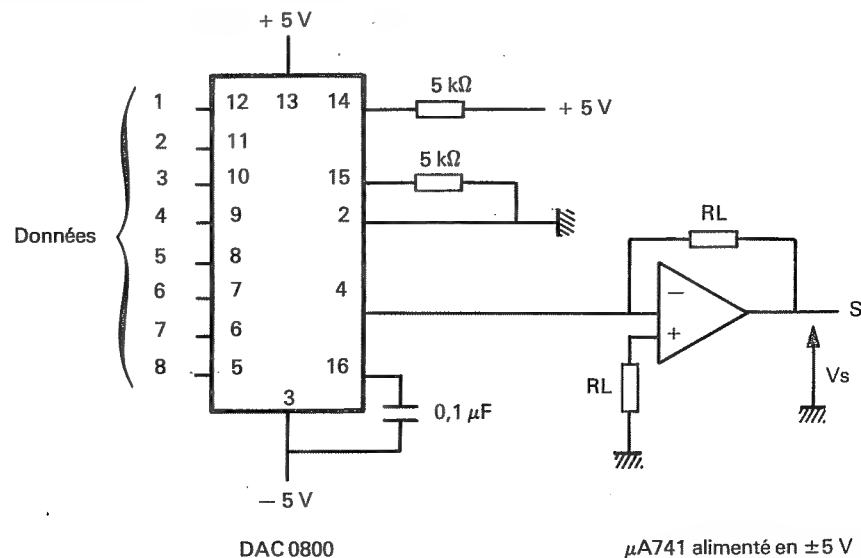


	1	NAME	PARITE
	2		
	3	; CALCUL ET INTRODUCTION DE LA PARITE	
	4		
2000	5	ORG	2000H
	6		
2000 214020	7	DEBUT: LXI	H, 2040H ; CHARGEMENT DU POINTEUR
2003 060F	8	MVI	B, 0FH ; CHARGEMENT DU COMPTEUR
2005 0E80	9	MVI	C, 80H ; CHARGE BIT DE PARITE
2007 7E	10	SUITE: MOV	A, M ; APPEL DE CARACTERE
2008 B1	11	ORA	C ; MISE A 1 BIT DE PARITE
2009 E20D20	12	JPO	BON ; IMPAIR ? ALLER A BON
200C 77	13	MOV	M, A ; PAIR ? RENVOI EN MEMOIRE
200D 23	14	BON: INX	H ; POINTEUR +1
200E 05	15	DCR	B ; COMPTEUR -1
200F C20720	16	JNZ	SUITE ; SI (B) # 0 BOUCLER
2012 CF	17	RST	1
	18		
2000	19	END	DEBUT

CONVERSION : DIGITAL-ANALOGIQUE ET GENERATEUR DE FONCTIONS

But : Réalisation d'un générateur de fonctions (principe).

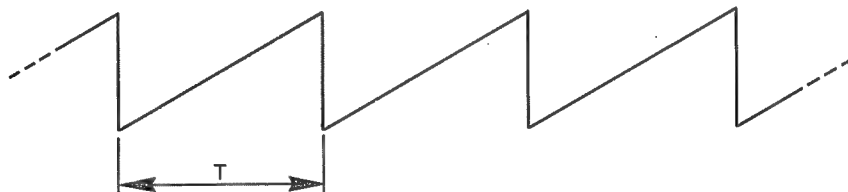
A l'aide d'un convertisseur digital-analogique 8 bits du type DAC 0800 et d'un amplificateur opérationnel du type $\mu A 741$, on peut réaliser un générateur de fonctions, c'est-à-dire un appareil qui délivre une certaine forme de signal plus ou moins répétitive. Pour cela il suffit de réaliser un montage dont le schéma est :



Nous aurons, pour R_L inférieur à $5k\Omega$, une valeur maximale de V_s égale à $5 R_L / 5000$, c'est-à-dire que pour $R_L = 1k\Omega$, on ne dépassera pas $1V$.

Ce circuit réalisé sera branché sur un port de la RAM (ou de la ROM) et le bit de poids faible du mot sortie vaudra ($V_s \text{ max} / 256$) Volts, c'est-à-dire que pour $R_L = 1k\Omega$, le bit de poids faible vaut $0,004V$ (soit $4mV$).

Pour réaliser une dent de scie, c'est-à-dire un signal de la forme :



nous écrirons le programme suivant, en supposant que la tension maximale du signal soit atteinte pour une donnée égale à FF :

```

1          .TITLE GENE
2
3          ;CONVERSION NUMERIQUE-ANALOGIQUE : GENERATEUR DE
4          ;FONCTIONS
5
6          05F1    DELAI    =005F1
7          AAAA    PER      =0AAAA          ;PERIODE DU SIGNAL
8                                         ;DIVISEE PAR 256
9
10 0000          . =02000
11
12 2000 3E01          MVI      A,001
13 2002 D320          OUT      020          ;PORT A=SORTIE
14 2004 31C020        LXI      SP,020C0
15 2007 32FF20        STA      020FF        ;POUR PAS A PAS
16 200A AF            XRA      A            ; (A)=0
17 200B D321    BOUCLE: OUT      021        ;ON SORT (A)
18 200D F5            PUSH     PSW
19 200E 11AAAA        LXI      D,PER
20 2011 CDF105        CALL     DELAI
21 2014 F1            POP      PSW
22 2015 3C            INR      A
23 2016 C30B20        JMP      BOUCLE
24
25          0000          .END

```

Ici, T est la période ; on charge donc la paire DE avec un nombre nous donnant un deux cent cinquante sixième de la période, comme on l'a vu dans un programme précédent.

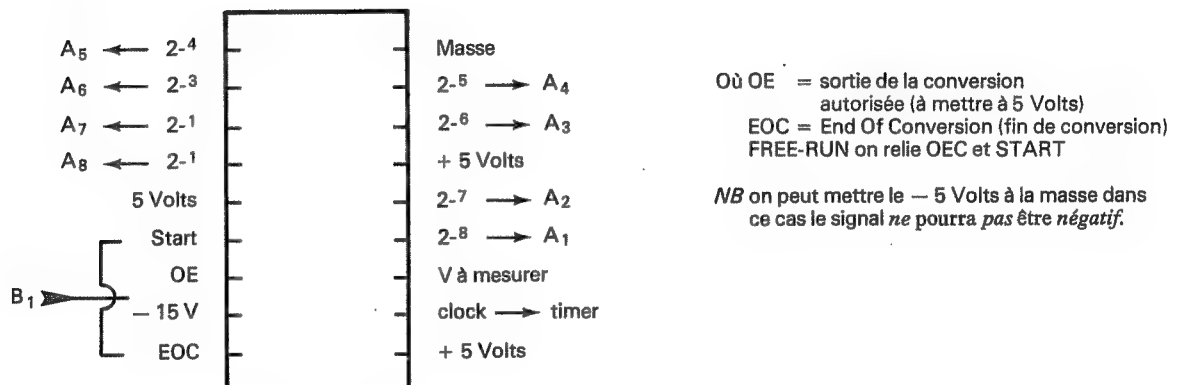
CONVERSION : ANALOGIQUE-DIGITAL ET VOLTMETRE DIGITAL

But : Principe d'un voltmètre numérique.

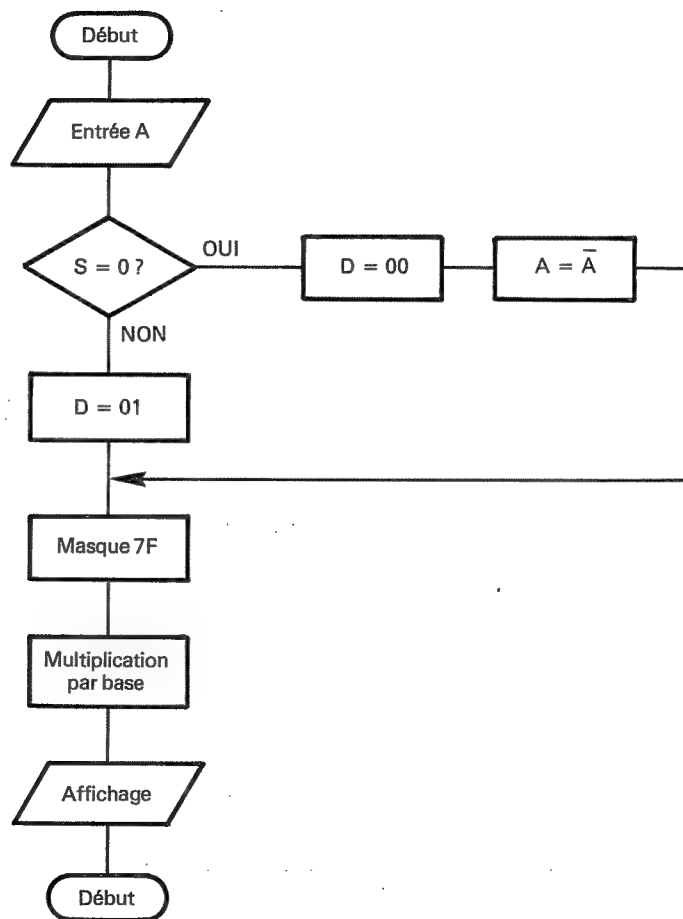
La conversion analogique-digital nécessite une horloge qui, dans le cas du circuit ADC 0800, doit être comprise entre 50 et 800 kHz. Cela impose l'emploi d'une deuxième RAM fournissant un timer. Il faut, en outre, alimenter le circuit de conversion avec trois tensions + 5, — 5, — 15 V si l'on désire mesurer un signal compris entre + et — 5 V.

Il faut fournir un signal de début de conversion, le circuit convertisseur prévenant (par interruption) quand la conversion est terminée. Mais on peut travailler en «roue libre», (free-run) la fin de conversion faisant partir une nouvelle conversion ; mais *il faut alors que le signal à mesurer varie peu* dans le temps puisque la conversion exige 40 périodes d'horloge c'est-à-dire à 500 kHz : 80 μ s.

Pour simplifier nous travaillons dans le deuxième cas. Le schéma du circuit ADC 0800 est le suivant :



Ce circuit donne en digital 00 pour + 5 V et FF pour — 5 V ; il faut donc écrire un programme qui détecte les valeurs positives et les valeurs négatives. L'organigramme est le suivant :



- D contient le signe 00 = + ; 01 = -
- La base vaut 40 mV (par simplification).

Le programme comporte deux parties, la première crée l'horloge à l'aide du timer de la deuxième RAM, la deuxième partie fait le calcul et l'affichage :

```

1          .TITLE  VOLTME,'TRE'
2
3          ;VOLTMETRE DIGITAL
4
5 0000          .==02000
6
7 2000 31C020      LXI      SP,020C0
8 2003 3E06        MVI      A,006          ;CHARGEMENT
9 2005 D32C        OUT      02C          ;DU TIMER
10 2007 3E40       MVI      A,040        ;A 0006
11 2009 D32D       OUT      02D          ; = 500 KHZ
12 200B 3EC2       MVI      A,0C2        ;DEMARRAGE DU TIMER
13 200D D328       OUT      028          ;ET PORT A =ENTREE
14                ;   PORT B =SORTIE
15 200F 3E01       MVI      A,001        ;BIT 0 (LE 1 ER) DU
16                ;PORT B = 1
17 2011 D32A       OUT      02A          ;AUTORISE LA SORTIE
18                ;DU CONVERTISSEUR

```

19	2013	DB29	DEBUT:	IN	029	#ENTRE DE LA VALEUR A
20						#AFFICHER
21	2015	B7		ORA	A	#POSITIONNE LES FLAGS
22	2016	1601		MVI	D,001	
23	2018	FA1D20		JM	NEG	
24	2018	15		DCR	D	#(D)=0 VALEUR POSITIVE
25	201C	2F		CMA		
26	201D	E67F	NEG:	ANI	07F	#LE MASQUE ELIMINE LE
27						#BIT DE SIGNE
28	201F	5F		MOV	E,A	
29	2020	2E40		MVI	L,040	#LA "BASE" EST DANS L
30	2022	010000		LXI	B,00000	#INITIALISATION
31	2025	7D	BOUCLE:	MOV	A,L	
32	2026	81		ADD	C	
33	2027	27		DAA		
34	2028	4F		MOV	C,A	
35	2029	78		MOV	A,B	#TRAITEMENT DES
36	202A	CE00		ACI	00	#CENTAINES, AJOUTE 1
37	202C	27		DAA		#A (B) SI CY=1
38	202D	47		MOV	B,A	
39	202E	1D		DCR	E	
40	202F	C22520		JNZ	BOUCLE	
41	2032	2619		MVI	H,019	#ADRESSE DU 8279 =19XX
42	2034	3690		MVI	M,090	
43	2036	25		DCR	H	
44	2037	7A		MOV	A,D	
45	2038	B7		ORA	A	
46	2039	C24120		JNZ	NEGA	
47	203C	3636		MVI	M,036	
48	203E	C34320		JMP	AFF	
49	2041	36FB	NEGA:	MVI	M,0FB	#FB= -
50	2043	116B20	AFF:	LXI	D,TABLE	
51	2046	6B		MOV	L,E	#(L) HAUT DE TABLE
52	2047	78		MOV	A,B	
53	2048	E6F0		ANI	0F0	#ISOLE LE QUARTET HAUT
54	204A	07		RLC		
55	204B	07		RLC		
56	204C	07		RLC		
57	204D	07		RLC		
58	204E	83		ADD	E	
59	204F	5F		MOV	E,A	#(DE) POINTE LA TABLE
60						#DE CONVERSION
61	2050	1A		LDAX	D	
62	2051	E6F7		ANI	0F7	#AJOUTE POINT DECIMAL
63						#UN 0 ALLUME
64	2053	77		MOV	M,A	#AFFICHE LES "VOLTS"
65	2054	5D		MOV	E,L	#(DE)="TABLE"
66	2055	78		MOV	A,B	
67	2056	E60F		ANI	00F	#ISOLE LE QUARTET BAS
68	2058	83		ADD	E	
69	2059	5F		MOV	E,A	
70	205A	1A		LDAX	D	
71	205B	77		MOV	M,A	#AFFICHE LES DIXIEMES
72	205C	5D		MOV	E,L	
73	205D	79		MOV	A,C	
74	205E	E6F0		ANI	0F0	

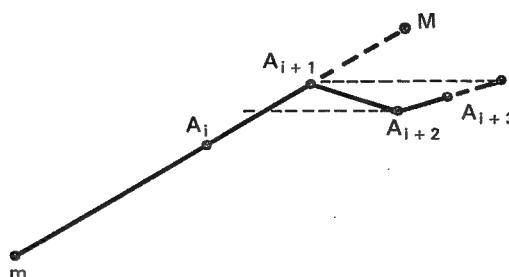
75	2060	07		RLC		
76	2061	07		RLC		
77	2062	07		RLC		
78	2063	07		RLC		
79	2064	83		ADD	E	
80	2065	5F		MOV	E, A	
81	2066	1A		LDAX	D	
82	2067	77		MOV	M, A	
83	2068	C31320		JMP	DEBUT	
84						
85	206B	0C	TABLE:	.BYTE	00C	; "0"
86	206C	9F		.BYTE	09F	; "1"
87	206D	4A		.BYTE	04A	; "2"
88	206E	0B		.BYTE	00B	; "3"
89	206F	99		.BYTE	099	; "4"
90	2070	29		.BYTE	029	; "5"
91	2071	2B		.BYTE	02B	; "6"
92	2072	8F		.BYTE	08F	; "7"
93	2073	0B		.BYTE	00B	; "8"
94	2074	89		.BYTE	089	; "9"
95						
96		0000		.END		

CONVERSION : ANALOGIQUE-DIGITAL PAR APPROXIMATIONS SUCCESSIVES

But : Examiner une autre méthode de conversion.

On peut remplacer un convertisseur analogique-digital par un convertisseur digital-analogique et un comparateur, le traitement pour arriver à la conversion analogique digitale étant assuré par un programme.

Principe.— On donne une valeur digitale qui est convertie en analogique, puis comparée à la valeur à convertir. En sortie du comparateur, on a un 0 ou un 1 selon que la valeur obtenue est supérieure ou inférieure à la valeur à convertir (ou vice-versa) ; on teste par exemple, l'entrée série (SID) pour savoir on l'on est. La nouvelle valeur digitale à convertir sera la moyenne arithmétique entre des valeurs précédemment essayées selon le schéma suivant, avec m = minimum et M = maximum. Le sens de variation dépend de la réponse au comparateur.



Essai	Réponse	Commentaire
m	trop petit	A_i remplace m
M	trop grand	
$A_i = \frac{m + M}{2}$	trop petit	
$A_{i+1} = \frac{A_i + M}{2}$	trop grand	A_{i+1} remplace M
$A_{i+2} = \frac{A_{i+1} + A_i}{2}$	trop petit	A_{i+2} remplace m
$A_{i+3} = \frac{A_{i+2} + A_{i+1}}{2}$	trop petit	

L'excursion des valeurs allant de 00 à FF, le premier essai sera fait avec 80 ; suivant la réponse, on tentera 40, ou BF, etc., selon le programme suivant :

```

7          .BYTE    020
8          .ENDM
9
10         034E    UPDDT    =0034E
11
12 0000          . =02000
13
14 2000 3E01      MVI     A,001
15 2002 D320      OUT     020          ;PORT A =SORTIE
16 2004 32FF20    STA     020FF       ;POUR PAS A PAS
17 2007 3E80      MVI     A,080       ;TEST VALEUR CENTRALE
18 2009 47        MOV     B,A         ;(B)=CETTE VALEUR
19 200A D321      OUT     021          ;SORTIE DE L'ESSAI
20 200C          RIM
21 200D 17        RAL
22 200E DA3420    JC      T60         ;SI CY=1 : TROP GRAND
23 2011 0E00      MVI     C,00
24 2013 3E40      MVI     A,040
25 2015 D321      BOUCLE: OUT     021
26 2017          RIM
27 2018 17        RAL
28 2019 DA2920    JC      T61
29 201C DB21      IN      021         ;VALEUR TESTEE RELUE
30 201E 47        MOV     B,A
31 201F 81        ADD     C
32 2020 B7        ORA     A           ;CY=0
33 2021 1F        RAR
34 2022 B8        CMP     B           ;(A)=(A)/2
35                                     ;ON COMPARE A LA PLUS
36                                     ;PETITE
36 2023 CA3D20    JZ      FIN
37 2026 C31520    JMP     BOUCLE
38 2029 DB21      T61:  IN      021
39 202B 4F        MOV     C,A         ;VALEUR ESSAYEE DANS C
40 202C 80        ADD     B
41 202D B7        ORA     A
42 202E 1F        RAR
43 202F B8        CMP     B
44 2030 CA3D20    JZ      FIN
45 2033 C31520    JMP     BOUCLE
46 2036 0EFF      T60:  MVI     C,0FF
47 2038 3EDF      MVI     A,0BF       ;BF=(FF+80)/2
48 203A C31520    JMP     BOUCLE
49 203D 31C020    FIN:  LXI     SP,020C0

```

Les valeurs supérieures sont stockées temporairement dans C, les valeurs inférieures dans B ; c'est pourquoi nous comparons la valeur à essayer à celle qui est dans B. Il ne faut pas oublier que nous travaillons à 1 bit près, ce qui veut dire que 39 peut être trop petit et que 3A sera trop grand.

Au premier «passage», il faut, au maximum 8 essais. Si l'on a fait une erreur entre JC et JNC, il en faudra 16. Si la grandeur à convertir varie lentement, il faudra peu d'essais pour la suivre.

Il faut remarquer que l'on a écrit IN port A alors que ce port est un port *de sortie* ; cela, parce que les sorties sont «latchées» c'est-à-dire mémorisées : il est donc possible de les lire sans les perturber.

TIR AU PIGEON

Le jeu consiste à arrêter, pour gagner, un tiret (affichage sur 7 segments) ou tout autre signe (travail avec écran de visualisation). Nous donnons ci-dessous le programme pour l'affichage 7 segments, l'arrêt est obtenu par l'intermédiaire de l'interruption RST 7,5. Le nombre de tirs à effectuer est rangé dans le registre C. Le nombre de points obtenus est affiché à la fin du jeu. On gagne 1 point à chaque tir si on arrête le tiret sur le troisième digit. Le microprocesseur avertit le tireur de son succès par affichage de «tch».

On peut modifier la vitesse du tiret. Remarquez que la commande d'extinction codée «CD» qui a pour effet le remplissage de la mémoire-affichage du 8279 (16 octets) avec FF, rend le circuit indisponible pendant $160 \mu s$ (à 3 MHz). On peut soit attendre un certain temps, soit ce qui est fait ici, tester l'état du circuit en lisant son mot d'état (*Status Word*) à l'aide de l'instruction LDA 1900 H, ou MOV A, M si H contient 19 H. Le bit de poids le plus fort du mot d'état est à 0 si le circuit est disponible.

```

1          .TITLE  TIR
2
3          #TIR AU PIGEON ,JEU DE REFLEXES
4
5          .MACRO  SIM
6          .BYTE  030
7          .ENDM
8
9          05F1    DELAI    =005F1
10         034E    UPDDT    =0034E
11
12 0000          . =02000
13
14 2000 010900    LXI      B,00009          #NBRE POINTS DANS B
15                                     #NBRE TIRS DANS C
16 2003 31C020 #5: LXI      SP,020C0
17 2006 3E1B      MVI      A,01B          #AUTORISE INTERRUPTIONS
18                                     #AVEC R.A.Z. DE RST7.5
19 200B          SIM
20 2009 FB        EI
21 200A 21B019 #2: LXI      H,019B0          # (L)=NUMERO DU DIGIT
22 200D 34CD      #1: MVI      M,0CD          #EXTINCTION
23 200F 7E        #0: MOV      A,M
24 2010 E6B0      ANI      0B0          #TEST BIT 7 DU MOT
25                                     #D'ETAT DU 8279
26 2012 C20F20    JNZ      #0
27 2015 75        MOV      M,L
28 2016 25        DCR      H
29 2017 34FB      MVI      M,0FB          #AFFICHE ~~~
30 2019 110CF0    LXI      D,0F000          #VITESSE DU TIRET

```

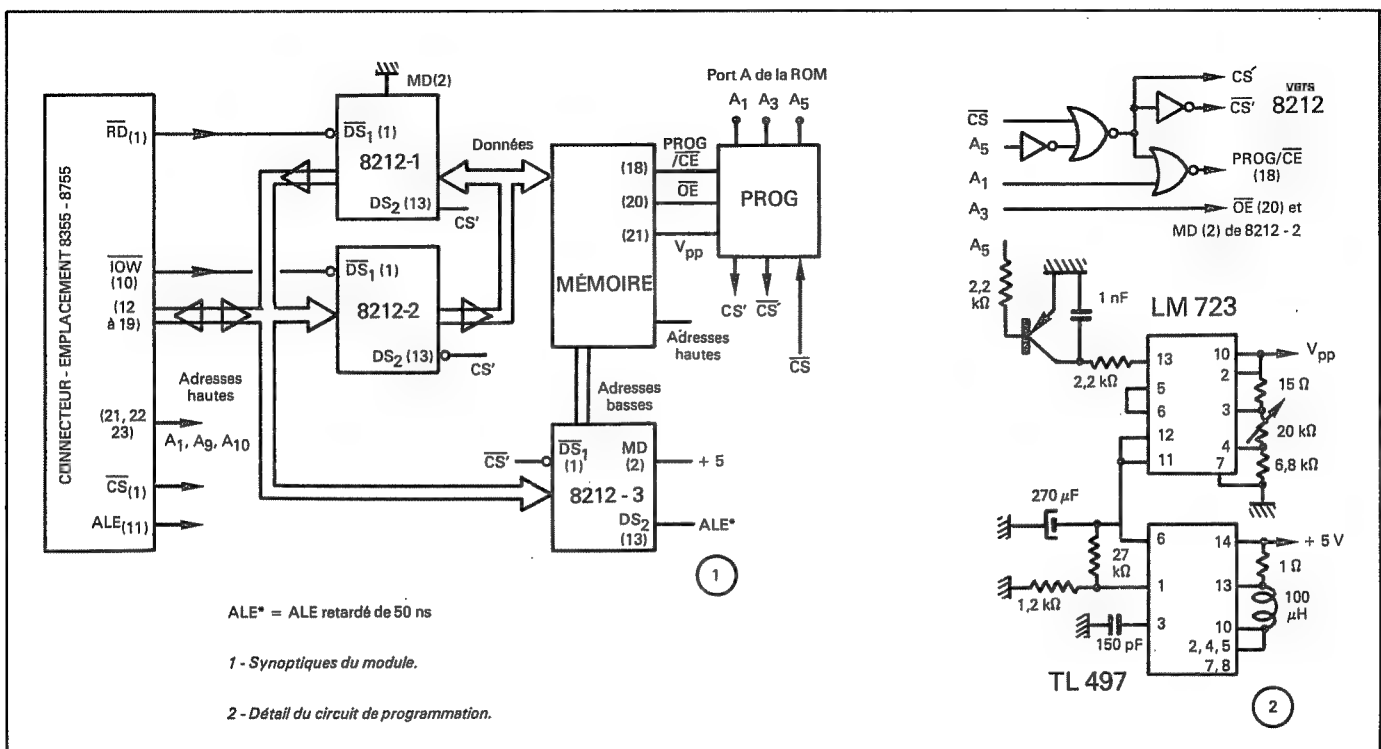
31	201C	CDF105	CALL	DELAI	
32	201F	24	INR	H	
33	2020	2C	INR	L	
34	2021	7D	MOV	A,L	
35	2022	FEB6	CPI	086	
36	2024	C20D20	JNZ	\$1	
37	2027	C30F20	JMP	\$0	
38					
39	202A	11FFFF INTR:	LXI	D,0FFFF	#TEMPO. ANTI REBONDS
40	202D	CDF105	CALL	DELAI	
41	2030	7D	MOV	A,L	
42	2031	FEB2	CPI	082	#TIR VALABLE ?
43	2033	C25420	JNZ	\$3	#NON, SAUT \$3
44	2036	04	INR	B	#OUI, NBRE POINTS + 1
45	2037	0D	DCR	C	#NBRE TIRS - 1
46	203B	C24020	JNZ	\$4	
47	203B	7B \$6:	MOV	A,B	
48	203C	CD6E03	CALL	UPDDT	#AFFICHE LE SCORE
49	203F	76	HLT		
50	2040	2619 \$4:	MVI	H,019	
51	2042	3690	MVI	M,090	
52	2044	25	DCR	H	
53	2045	36F8	MVI	M,0FB	# "+"
54	2047	366C	MVI	M,06C	# "F"
55	2049	369B	MVI	M,09B	# "0"
56	204B	11FFFF	LXI	D,0FFFF	
57	204E	CDF105	CALL	DELAI	
58	2051	C30320	JMP	\$5	
59	2054	0D \$3:	DCR	C	#NBRE TIRS - 1
60	2055	CA3B20	JZ	\$6	
61	205B	C30320	JMP	\$5	#REPREND AU DEBUT
62					
63	205B		==020CE		
64					
65	20CE	C32A20	JMP	INTR	
66					
67		0000	.END		

PROGRAMMATEUR DE MEMOIRE EPROM

Certain d'entre vous seront intéressés par la réalisation d'un programmeur de mémoire EPROM du type 2708 (tri-tension) ou 2716 (mono-tension). La capacité de la première est de 1 K octets (8 K bits) celle de la deuxième de 2 K octets. Les principes de programmation diffèrent suffisamment pour nécessiter deux programmes dont les listages vous sont donnés ci-après.

On utilise l'emplacement de la deuxième mémoire morte du kit SDK 85 ce qui nous fournit tous les signaux utiles (adresses, données, \overline{RD} , \overline{WR} , ALE, \overline{CS}). Le contrôle du programmeur est assuré par l'intermédiaire des ports de la ROM-moniteur. Ce programmeur permet de tester le programme avant l'écriture en EPROM puisqu'il accepte les RAM 2 K octets compatibles 2716. Vous apporterez à votre kit une plus grande souplesse de travail en programmant une EPROM avec le programme d'entrée-sortie magnétophone qui est fourni par INTEL dans le USER'S MANUAL du 8085.

Synoptique au programmeur



Les adresses hautes sont stabilisées, en cours de programmation, à l'aide de cavaliers.

```

1          .TITLE  VIRGIN,'ITE'
2
3          #TEST DE VIRGINITE AVANT PROGRAMMATION
4
5          FFFF          ADR          =0FFFF
6          0363          UPDAD        =00363
7          05F1          DELAY        =005F1
8
9          0000          . =02000
10
11
12 2000 31C020          LXI          SP,020C0
13 2003 21FFFF          LXI          H,ADR          #ADRESSE PREMIER OCTET
14                                     #PARTIE A TESTER
15 2006 7E          #1:          MOV          A,M
16 2007 FEFF          CPI          OFF
17 2009 C21420          JNZ          $0
18 200C 23          #2:          INX          H
19 200D 7C          MOV          A,H
20 200E FE0C          CPI          00C          #1K OCTETS OU 10 FOUR
21                                     #2K OCTETS
22 2010 C20620          JNZ          $1
23 2013 CF          RST          1
24 2014 E5          #0:          PUSH        H
25 2015 EB          XCHG
26 2016 CD6303          CALL          UPDAD          #AFFICHE ADRESSE NON
27                                     #VIERGE
28 2019 0402          MVI          B,002
29 201B 11FFFF          #3:          LXI          D,0FFFF
30 201E CDF105          CALL          DELAY          #PENDANT 1 SECONDE
31 2021 05          DCR          B
32 2022 C21B20          JNZ          $3
33 2025 E1          POP          H
34 2026 C30C20          JMP          $2
35
36          0000          .END

```

```

1          .TITLE  COMPAR
2
3          #TEST DE COMPARAISON APRES PROGRAMMATION
4
5          FFFF          ADRAM        =0FFFF
6          EEEE          ADPR0M       =0EEEE
7          0363          UPDAD        =00363
8          05F1          DELAY        =005F1
9          00FF          LMAX         =OFF
10
11          0000          . =02000
12
13 2000 31C020          LXI          SP,020C0
14 2003 21EEEE          LXI          H,ADPR0M          #ADRESSE PREMIER OCTET
15                                     #EN EPROM
16 2006 01FFFF          LXI          B,ADRAM          #ADRESSE PREMIER OCTET
17                                     #EN RAM
18 2009 0A          #1:          LDAX        B
19 200A BE          CMP          H
20 200B C21720          JNZ          $0

```

```

21 200E 0C      $2:   INR      C
22 200F 23      INX      H
23 2010 7D      MOV      A,L
24 2011 FEFF    CPI      LMAX      ;LMAX=ADRESSE +1
25                                     ;DERNIER OCTET EPROM
26 2013 C20920    JNZ      $1
27 2014 CF      RST      1
28 2017 E5      $0:   PUSH    H
29 2018 05      PUSH    B
30 2019 EB      XCHG
31 201A CD4303    CALL     UPDAD     ;AFFICHE ADRESSE CASE
32                                     ;NON CONFORME
33 201D 0602      MVI      B,002
34 201F 11FFFF    $3:   LXI      D,0FFFF
35 2022 CDF105    CALL     DELAY
36 2025 05      DCR      B
37 2026 C21F20    JNZ      $3
38 2027 C1      POP      B
39 202A E1      POP      H
40 202B C30E20    JMP      $2
41
42      0000      .END

```

```

1      .TITLE   PROGMO,"NO"
2
3      ;PROGRAMMATION DES 2758/2716 MONOTENSION
4
5      FFFF      ADRAM    =0FFFF
6      08EE      ADFROM   =008EE
7      034E      UPDDT    =0034E
8      05F1      DELAY    =005F1
9      0002      DDRA     =002
10     0000      PORTA    =000
11     00FF      NOMB     =0FF
12
13 0000      . =02000
14
15 2000 31C020    LXI      SP,020C0
16 2003 21EE08    LXI      H,ADFROM      ;ADRESSE PREMIER OCTET
17                                     ;EN EPROM,PARTIE BASSE
18                                     ;DANS L ,PARTIE HAUTE
19                                     ;PAR CAVALIERS
20 2006 01FFFF    LXI      B,ADRAM      ;ADRESSE PREMIER OCTET
21                                     ;EN RAM
22 2009 3EFF      MVI      A,NOMB      ;LONGUEUR DU PROGRAMME
23                                     ;256 OCTETS = 00!(100H)
24 200B F5      PUSH    PSW
25 200C 3EFF      MVI      A,0FF      ;C.W. PORT A DE ROM
26 200E D302      OUT     DDRA
27 2010 3E28      $1:   MVI      A,028
28 2012 D300      OUT     PORTA
29 2014 0A      LDAX    B
30 2015 77      MOV      M,A
31 2016 3E0A      MVI      A,00A
32 2018 D300      OUT     PORTA      ;VPP A 25 VOLTS
33 201A 11F000    LXI      D,000F0    ;TEMPO 1,5 MS
34 201D CDF105    CALL     DELAY

```



```

35 2020 3E08      MVI      A,008
36 2022 D300      OUT      PORTA
37 2024 11F01B    LXI      D,01BF0      #TEMPO 50 MS
38 2027 CDF105    CALL     DELAY
39 202A 3E0A      MVI      A,00A
40 202C D300      OUT      PORTA
41 202E 11F000    LXI      D,000F0      #TEMPO 1,5 MS
42 2031 CDF105    CALL     DELAY
43 2034 3E20      MVI      A,020
44 2036 D300      OUT      PORTA      #VPP A 0 VOLT
45 2038 0C        INR      C
46 2039 2C        INR      L
47 203A CA4720    JZ       $0          #ARRET TEMPORAIRE
48 203D F1        #2:      POP     PSW
49 203E 3D        DCR      A          #NOMB = NOMB - 1
50 203F F5        PUSH    PSW
51 2040 C21020    JNZ      $1          #ON CONTINUE
52 2043 CD4E03    CALL     UPDDT      #SINON ON AFFICHE 00
53 2046 76        HLT
54 2047 CF        #0:      RST       1      #AFFICHE 8085 ARRET
55                                     #TEMPORAIRE ,CHANGER
56                                     #LES CAVALIERS ADRES.
57                                     #HAUTE. ON REPART PAR
58                                     # 80 EXEC
59 2048 C33D20    JMP      $2
60
61      0000      .END

1      .TITLE  PROGTR
2
3      #PROGRAMMATION 2708 TRI-TENSION
4
5      FFFF      ADRAM   =0FFFF
6      08EE      ADPR0M  =008EE
7      036E      UPDDT   =0036E
8      05F1      DELAY   =005F1
9      0002      DDRA    =002
10     0000      PORTA   =000
11     FF00      NOMB    =0FF00
12
13 0000      . =02000
14
15 2000 31C020    LXI      SP,020C0
16 2003 21EE08    LXI      H,ADPR0M      #ADRESSE PREMIER OCTET
17                                     #EN EPROM, PARTIE BASSE
18                                     #DANS L ,PARTIE HAUTE
19                                     #PAR CAVALIERS
20 2006 01FFFF    LXI      B,ADRAM      #ADRESSE PREMIER OCTET
21                                     #EN RAM
22 2009 1100FF    LXI      D,NOMB      # (D) = NOMB = LONGUEUR
23                                     #DU PROGRAMME ,256= 00
24 200C 3EFF      MVI      A,0FF      #C.M. PORT A DE LA ROM
25 200E D302      OUT      DDRA
26 2010 7D        MOV      A,L
27 2011 2F        CMA
28                                     # (L) COMPLEMENTE
28 2012 3C        INR      A
29 2013 BA        CMP      D          #COMPARE A NOMB
30 2014 D21C20    JNC      $0          #PLACE DISPONIBLE ?
31                                     #SAUT SI L=FF

```

32	2017	5A		MOV	E,D	
33	2018	57		MOV	D,A	#(D) = NBRE D'OCTETS
34						#AVANT L=FF
35	2019	7B		MOV	A,E	
36	201A	B5		ADD	L	
37	201B	5F		MOV	E,A	#(E)=NBRE D'OCTETS
38						#AU DELA DE L=FF
39	201C	3E80	#0:	MVI	A,080	#NBRE BOUCLES A FAIRE
40	201E	D5	#2:	PUSH	D	
41	201F	C5		PUSH	B	
42	2020	E5		PUSH	H	
43	2021	F5		PUSH	PSW	
44	2022	3E28		MVI	A,028	
45	2024	D300		OUT	PORTA	
46	2026	D5	#1:	PUSH	D	#SALVEGARDE NOMB
47	2027	0A		LDAX	B	
48	2028	77		MOV	M,A	
49	2029	111000		LXI	D,00010	
50	202C	CDF105		CALL	DELAY	#TEMPORISATION
51	202F	3E08		MVI	A,008	
52	2031	D300		OUT	PORTA	#VPP A 26 VOLTS
53	2033	117D00		LXI	D,0007D	#PENDANT 1 MS
54	2036	CDF105		CALL	DELAY	
55	2039	3E28		MVI	A,028	
56	203B	D300		OUT	PORTA	
57	203D	111000		LXI	D,00010	
58	2040	CDF105		CALL	DELAY	
59	2043	0C		INR	C	
60	2044	2C		INR	L	
61	2045	D1		POP	D	
62	2046	15		DCR	D	#NOMB = NOMB - 1
63	2047	C22620		JNZ	#1	
64	204A	F1		POP	PSW	
65	204B	E1		POP	H	
66	204C	C1		POP	B	
67	204D	D1		POP	D	
68	204E	3D		DCR	A	#DECOMPTE LES BOUCLES
69	204F	C21E20		JNZ	#2	
70	2052	7B		MOV	A,E	
71	2053	B7		ORA	A	#DOIT ON FAIRE UNE
72						#2 EME PASSE ? PROG.
73						#AU DELA DE (L)=FF
74	2054	CA6220		JZ	FIN	
75	2057	CF		RST	1	#AFFICHE -8085 POUR
76						#CHANGEMENT CAVALIERS
77						#REPART PAR GO EXEC.
78	2058	7A		MOV	A,D	
79	2059	B1		ADD	C	
80	205A	4F		MOV	C,A	#NOUVELLE VALEUR DE C
81	205B	53		MOV	D,E	#NOUVELLE VALEUR DE D
82	205C	AF		XRA	A	#(A)=00
83	205D	5F		MOV	E,A	
84	205E	4F		MOV	L,A	
85	205F	C31C20		JMP	#0	
86	2062	CD6E03	FIN:	CALL	UPDDT	#FIN AFFICHE 00
87	2065	76		HLT		
88						
89		0000		.END		

LE KIT SDK 85

Nous donnons ci-après les caractéristiques et le mode d'emploi du kit SDK 85. Les schémas vous seront utiles si vous désirez construire votre propre micro-ordinateur. En particulier le schéma de câblage du périphérique programmable 8279 qui est ici sous-employé puisqu'il peut gérer 16 afficheurs 7 segments et un clavier de 64 touches sans compter CNTRL et SHIFT correspondant aux deux bits de poids fort du code des touches... vous pouvez donc l'utiliser pour vous faire un beau piano.

LE MATERIEL : LE MICROPROCESSEUR 8085 ET LE SDK-85

Composition du SDK-85

Le micro-ordinateur dont l'utilisation est proposée ici est le SDK-85, de *Intel*. Il est considéré comme le micro-ordinateur servant à apprécier le microprocesseur 8085, à mots de 8 bits, et a été commercialisé sous forme de kit. Son montage constituera toujours l'un des meilleurs exercices préliminaires pour les électroniciens.

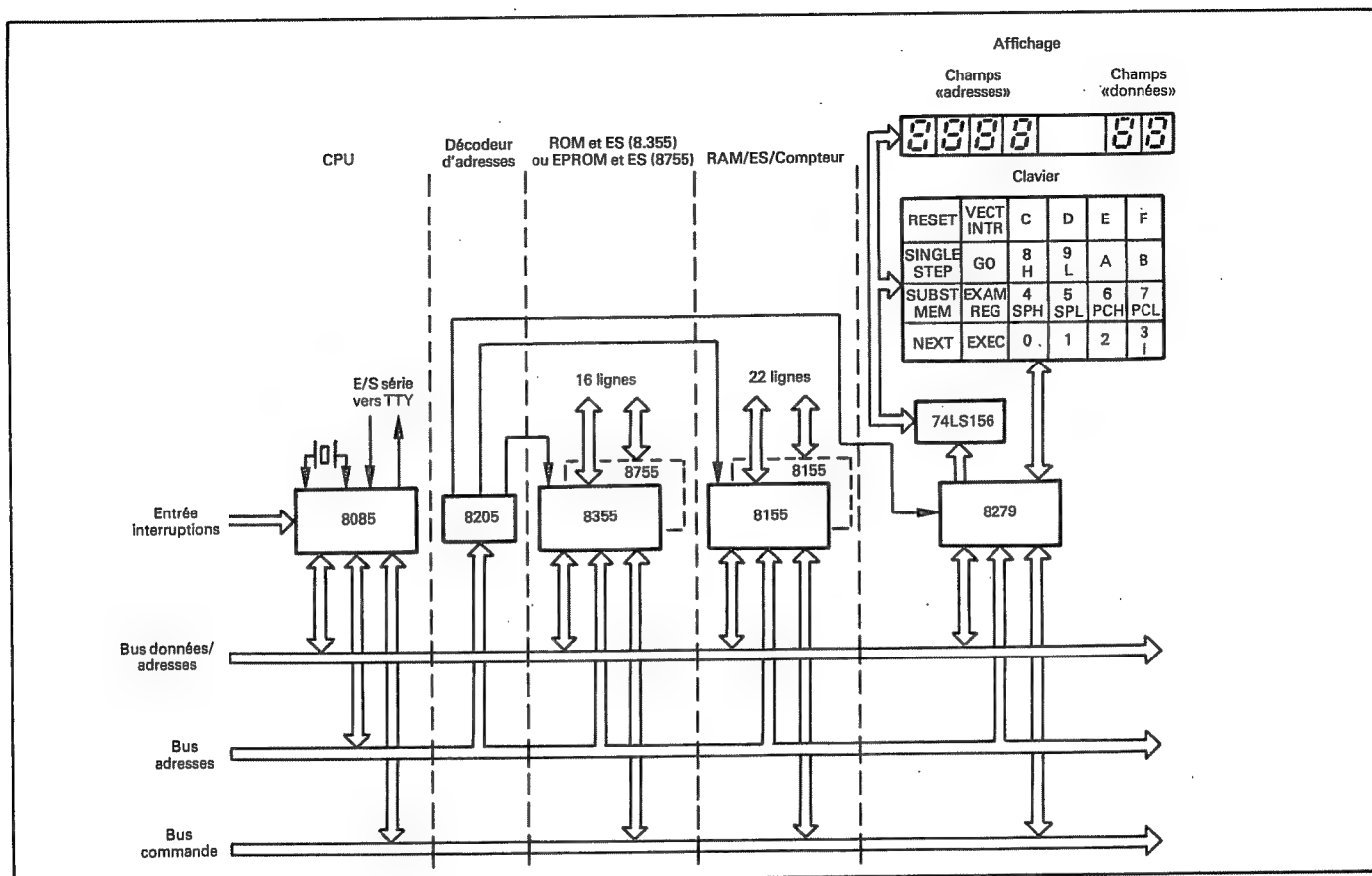
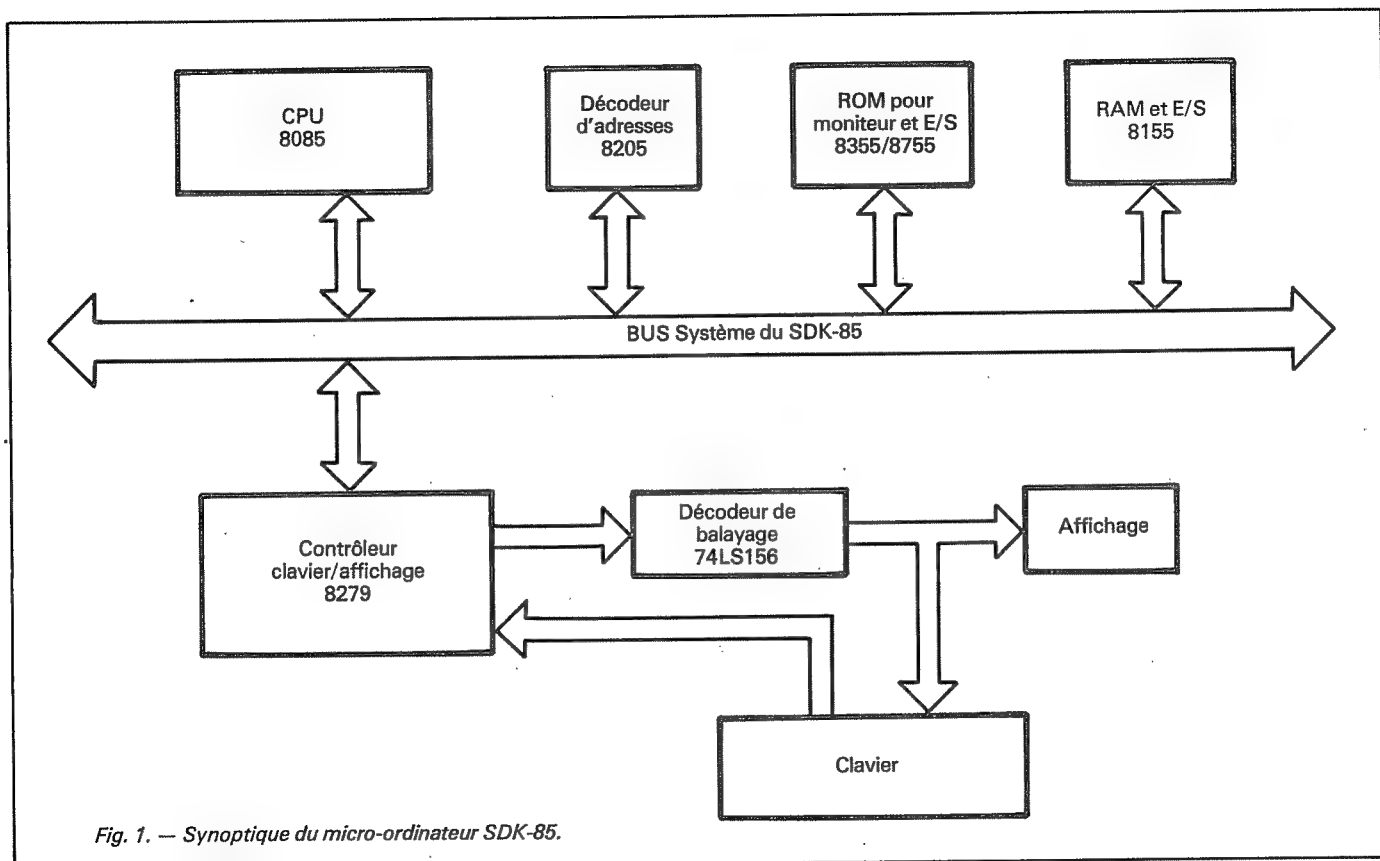
Le SDK-85 se compose (fig. 1, et 2 en plus détaillé) :

- Du microprocesseur 8085, à horloge de 3 MHz, soit un cycle d'exécution d'une addition de $1,3 \mu s$ (microcycle de 330 ns).
- De 2 K octets de mémoire morte, type 8355 ou 8755, contenant *le moniteur*.

Le moniteur est un programme de service, ou plutôt regroupe un ensemble de petits programmes qui exécutent des tâches fréquentes et en débarrassent donc l'utilisateur. Par exemple, le moniteur mettra en position de départ le micro-ordinateur dès sa mise sous tension ; c'est lui qui surveillera le clavier, etc.

- De 256 octets de RAM : c'est le circuit 8155. La RAM, tout comme la ROM, est éventuellement expansible, à 512 octets pour la première, et à 4 K octets pour la seconde, le tout sur la même carte.
- De 38 lignes d'entrées-sorties parallèles.
- D'une entrée-sortie série, via les ports SID/SOD du 8085.
- D'un clavier de 16 touches hexadécimales auxquelles s'ajoutent 8 touches de fonction.
- D'un affichage sur 6 digits à 7 segments, à diodes électroluminescentes.
- D'un circuit d'interface 8279 pour le clavier et l'affichage, associé à quelques composants discrets.

Le tout est monté sur une carte de 25×30 cm environ, dont la moitié est disponible pour les adjonctions à la demande. La consommation est de 0,5 A sous 5 V ; s'y ajoutera, si l'on connecte la carte à un téléimprimeur, du — 10 V (0,3 A). Le schéma électrique complet du SDK-85 est donné figure 3 pour le clavier et l'affichage, et 4 pour le CPU et les mémoires, RAM et ROM supplémentaires comprises.



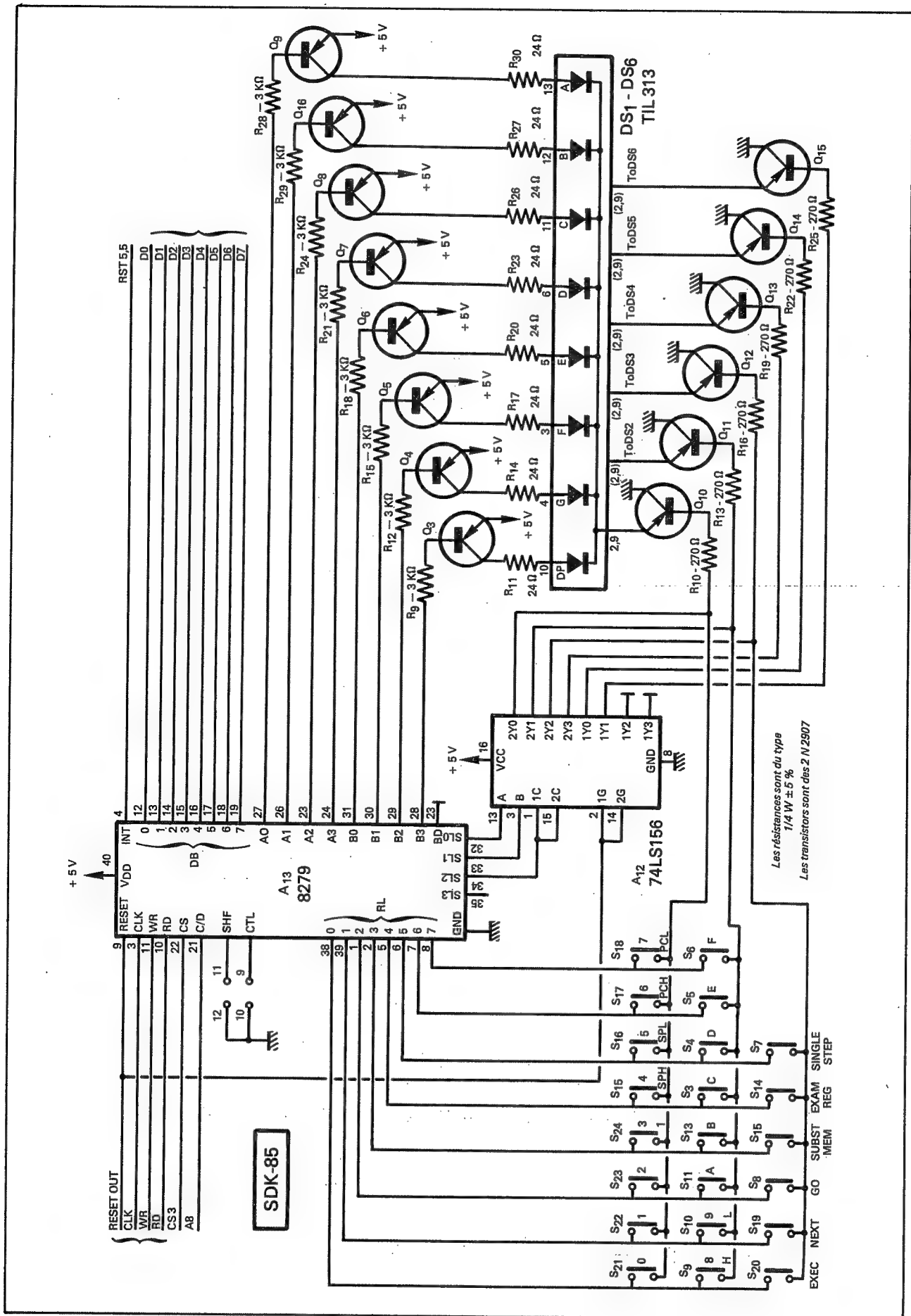
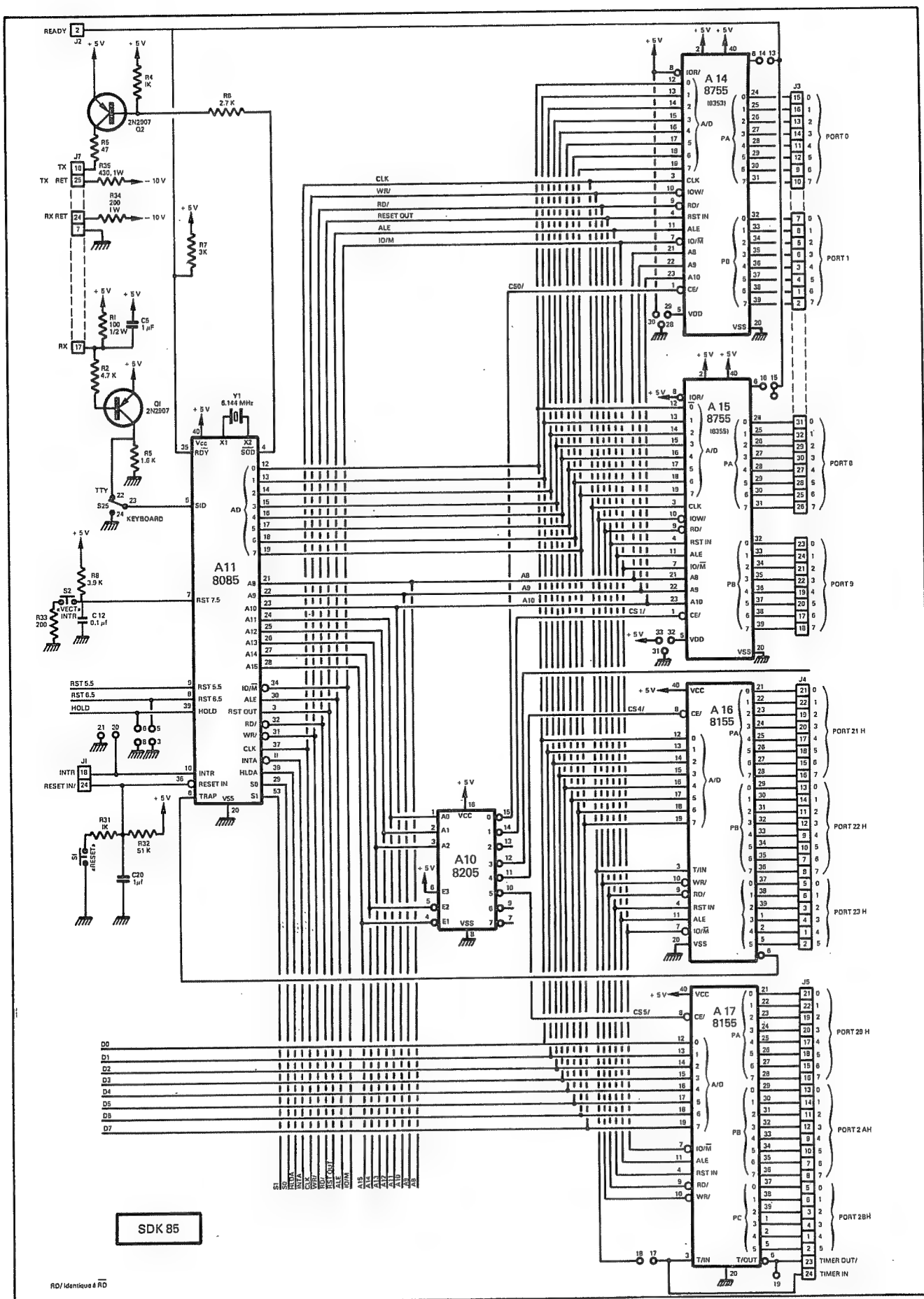


Fig. 3. — Schéma électrique du SDK-85: le clavier et l'affichage, et leurs commandes.



Mise en service

La carte est connectée à son alimentation, et mise sous tension. En appuyant sur le bouton RESET, qui correspond à la remise à zéro, on doit lire «— 8085» sur les afficheurs. La carte est désormais prête à entrer en action.

L'organisation des mémoires

Une mémoire se divise en cellules ; chaque cellule est dotée de sa propre adresse et contient une donnée, généralement sur 8 bits avec les microprocesseurs courants. C'est ce que montre la figure 5, où les notations sont en hexadécimal. Les contenus des cellules sont arbitraires.

Le décodage des adresses, tel qu'il a été prévu par câblage sur la carte (et grâce au circuit décodeur 8205) a attribué les adresses suivantes dont le respect est impératif :

— Le moniteur est logé en ROM aux adresses hexadécimales 0000 à 07FF incluses. Par conséquent, on peut seulement appeler ces adresses, ou lire leur contenu. Mais on n'a pas le droit de tenter d'y enregistrer quelque chose, ce qui est d'ailleurs impossible.

Si l'on tentait cette opération, le moniteur (qui a l'œil à tout) préviendrait l'utilisateur en commandant l'affichage d'un message d'erreur, soit «— Err».

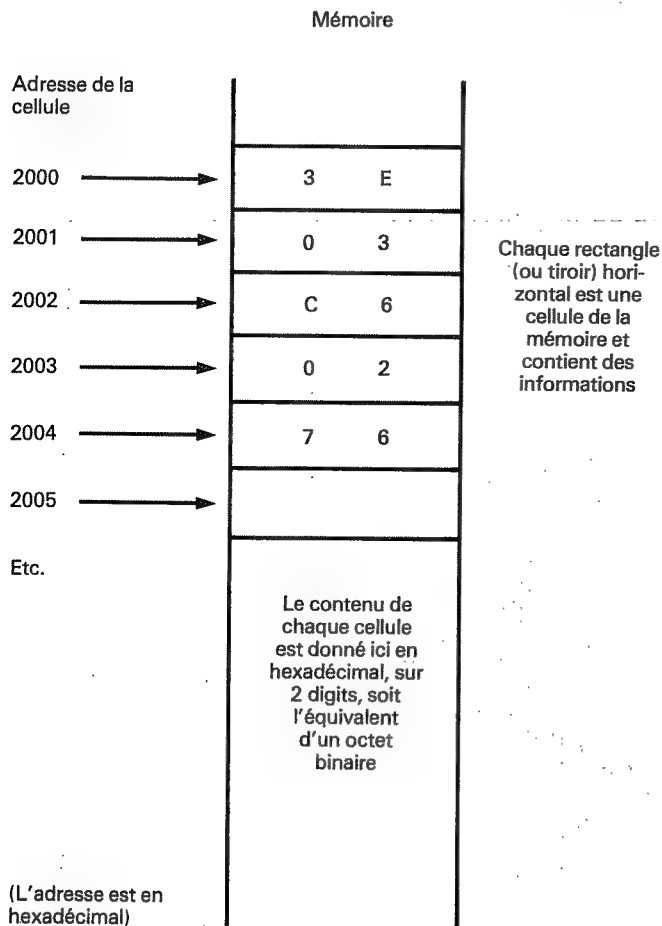


Fig. 5. — L'adresse et le contenu d'une cellule sont deux choses distinctes.

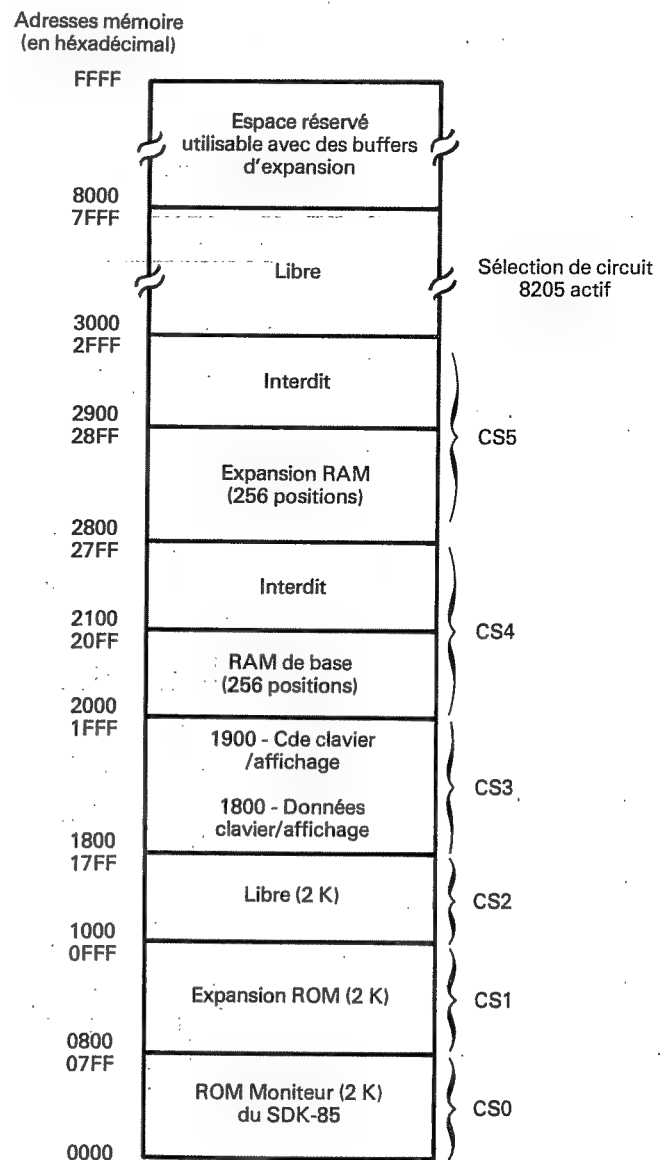


Fig. 6. — L'organisation de l'espace mémoire du SDK-85

— Les données (variables) sont stockées dans 256 octets de RAM aux adresses 2000 à 20FF incluses, toujours en hexadécimal.

La figure 6 montre l'organisation de l'espace-mémoire possible, qui va de 0000 à FFFF puisque, avec ses 16 bits d'adresse, le 8085 peut adresser jusqu'à 65536 cellules.

Ainsi donc, tous les programmes, et par conséquent tous les exercices proposés, sont stockés à partir de 2 000 (hexadécimal), valeur facile à retenir. Cependant, les adresses 20C0 à 20FF de la RAM ont été réservées à des besoins du moniteur ; l'exécution d'un programme, même en pas à pas, devra alors être précédée par le chargement de l'adresse 20C0 dans le pointeur de pile (ou une adresse inférieure) pour ne pas interférer avec le moniteur. On notera cependant que plusieurs moniteurs s'étant succédés, il faudra vérifier pour chacun d'eux (et on verra comment) quelles sont les adresses réellement réservées.

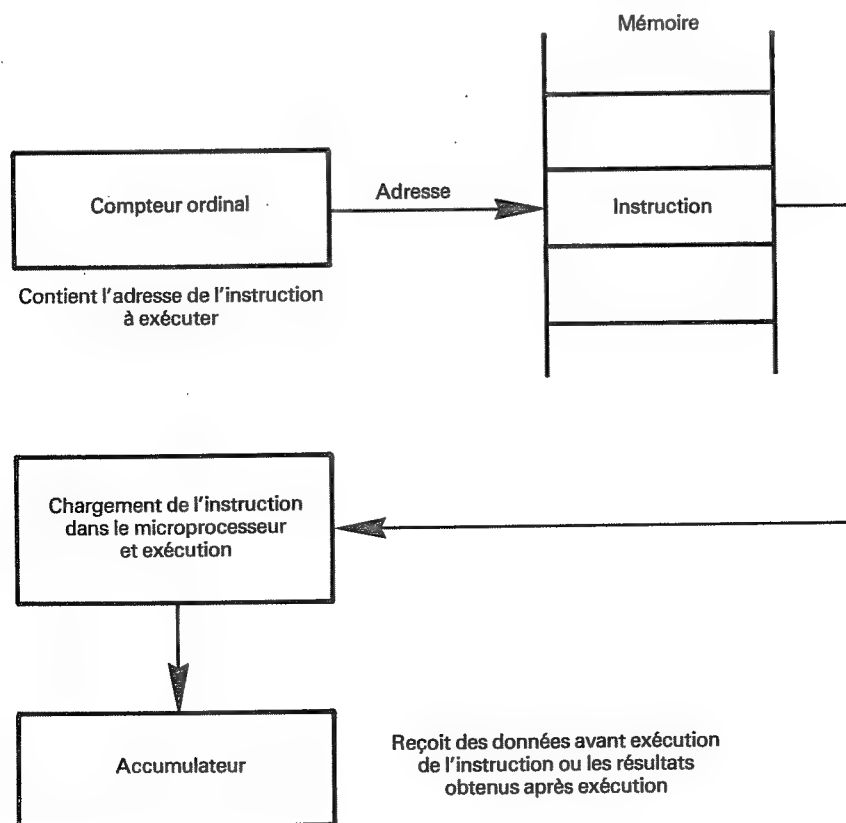


Fig. 7 - Un registre appelé compteur ordinal fournit l'adresse de la cellule-mémoire contenant la prochaine instruction à exécuter. Cette dernière est recopiée dans le microprocesseur puis exécutée. L'accumulateur contiendra des données diverses ou des résultats d'opérations s'il y a lieu.

Fonctionnement de base d'un microprocesseur

Rappelons maintenant comment procède le microprocesseur pour exécuter *un programme*. Un programme se compose d'une *liste d'instructions*. C'est à une petite mémoire contenue dans le microprocesseur, le *compteur ordinal*, qu'est confiée la mission de tenir à jour l'état instantané du déroulement du programme : le compteur ordinal contient *toujours* l'adresse de la cellule-mémoire où se trouve la prochaine instruction à exécuter. Par conséquent, on peut illustrer l'exécution d'une instruction simple comme le montre la figure 7 :

- Le compteur ordinal adresse la mémoire.
- Le contenu de la cellule adressée est transcrit dans le microprocesseur (*mais la cellule garde intact son contenu* qui pourra resservir). L'instruction est exécutée par le microprocesseur.

- Une autre petite mémoire, appelée *accumulateur*, reçoit des données ou des résultats (si, besoins est) avant ou après exécution de l'instruction.
- Au cours de l'exécution, le compteur ordinal est *automatiquement* incrémenté et pointe l'instruction suivante. Le programmeur n'a donc pas à se soucier de cette incrémentation ; il pourra cependant vérifier qu'elle suit bien le programme qu'il a fixé à la machine !

Encore un point mérite l'attention du lecteur : les cellules de la mémoire du SDK-85 peuvent contenir un octet (8 bits, ou encore deux digits hexadécimaux) ; c'est d'ailleurs le cas avec la plupart des microprocesseurs. Or, l'instruction peut exiger plus de 8 bits, 16 bits par exemple, ou même 24. Dans ce cas, on utilisera non plus une cellule, mais deux ou trois consécutives. Le premier code contenu dans la première d'une instruction informe le microprocesseur de sa longueur réelle ; la suite des opérations se fera donc à nouveau automatiquement.

Les instructions du 8085

La structure complète de principe du microprocesseur 8085 est donnée figure 8. On y notera la présence de multiples registres, dont les registres B, C, D, E, H et L mis à la disposition de l'utilisateur. Ce sont des registres de 8 bits chacun, qui peuvent être assemblés par paire : BC, ou DE, ou HL, pour former des ensembles 16 bits.

Le microprocesseur 8085 dispose des instructions résumées dans les deux tableaux suivants. Le premier donne leur mnémonique, leur définition, leur code d'instruction en binaire, et le nombre de cycles d'horloge qu'ils exigent pour exécution. Les lettres D et S des codes binaires doivent être remplacées par le code des registres *destinaire* ou *source*, comme indiqué.

Si l'on transpose ces codes binaires en hexadécimal, et si on classe les instructions par ordre des grandeurs hexadécimales, on obtient le second tableau.

Dans les exercices proposés ici, de nombreuses instructions sont utilisées ; elles ont été présentées au fur et à mesure des besoins, avec leurs incidences éventuelles. Une étude complète de ce jeu ne pourra se faire qu'avec la manuel d'utilisation du microprocesseur, en particulier pour ce qui concerne leur action sur les indicateurs (ou «Flags»).

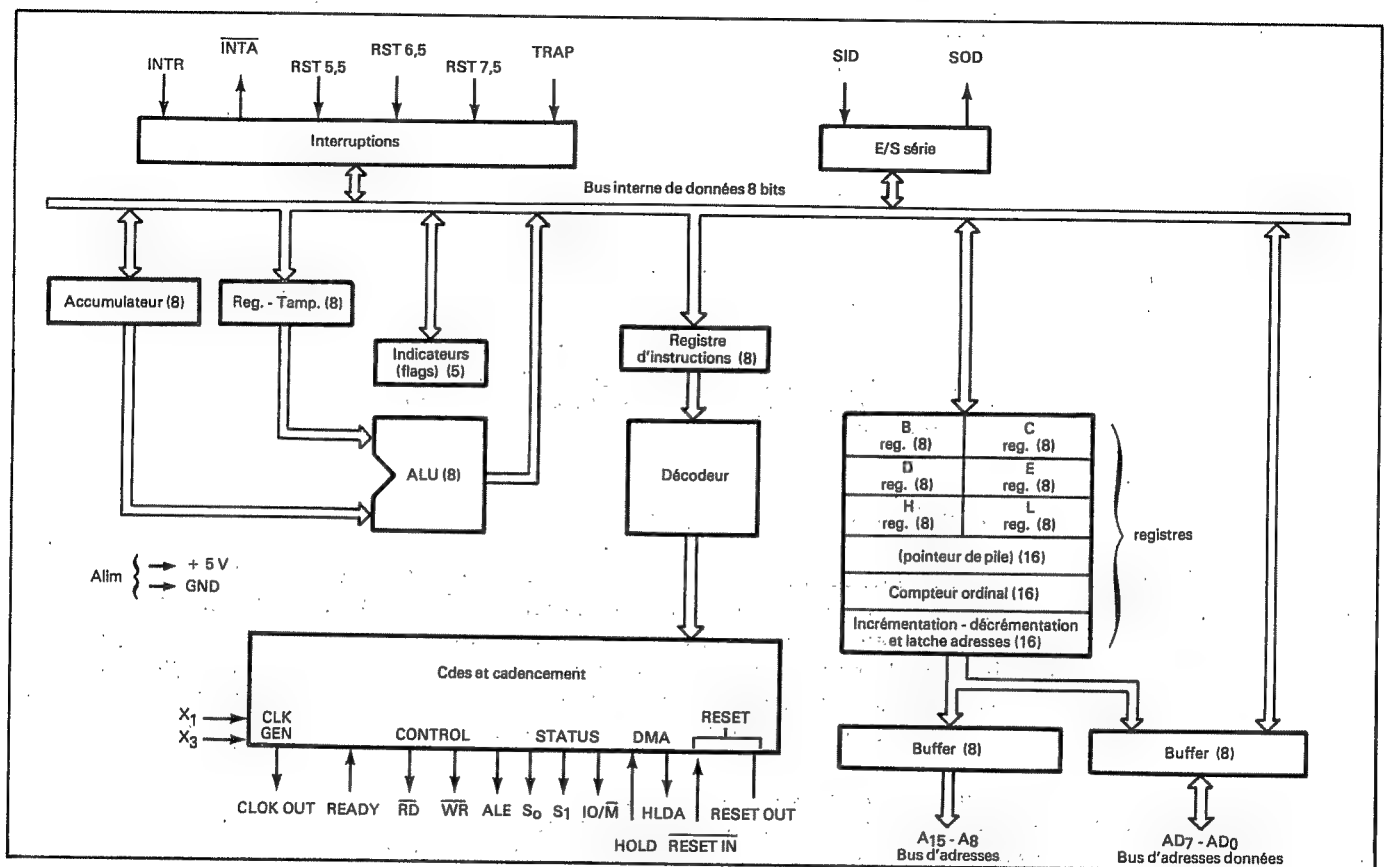


Fig. 8 - Architecture du 8075

Le jeu d'instructions du 8085 avec code binaire

Mnemonic	Description	Instruction Code (1)								Clock (2) Cycles
		D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀	
MOV r ₁ , r ₂	Move register to register	0	1	D	D	D	S	S	S	4
MOV M, r	Move register to memory	0	1	1	1	0	S	S	S	7
MOV r, M	Move memory to register	0	1	D	D	D	1	1	0	7
HLT	Halt	0	1	1	1	0	1	1	0	5
MVI r	Move immediate register	0	0	D	D	D	1	1	0	7
MVI M	Move immediate memory	0	0	1	1	0	1	1	0	10
INR r	Increment register	0	0	D	D	D	1	0	0	4
DCR r	Decrement register	0	0	D	D	D	1	0	1	4
INR M	Increment memory	0	0	1	1	0	1	0	0	10
DCR M	Decrement memory	0	0	1	1	0	1	0	1	10
ADD r	Add register to A	1	0	0	0	0	S	S	S	4
ADC r	Add register to A with carry	1	0	0	0	1	S	S	S	4
SUB r	Subtract register from A	1	0	0	1	0	S	S	S	4
SBB r	Subtract register from A with borrow	1	0	0	1	1	S	S	S	4
ANA r	And register with A	1	0	1	0	0	S	S	S	4
XRA r	Exclusive Or register with A	1	0	1	0	1	S	S	S	4
ORA r	Or register with A	1	0	1	1	0	S	S	S	4
CMP r	Compare register with A	1	0	1	1	1	S	S	S	4
ADD M	Add memory to A	1	0	0	0	0	1	1	0	7
ADC M	Add memory to A with carry	1	0	0	0	1	1	1	0	7
SUB M	Subtract memory from A	1	0	0	1	0	1	1	0	7
SBB M	Subtract memory from A with borrow	1	0	0	1	1	1	1	0	7
ANA M	And memory with A	1	0	1	0	0	1	1	0	7
XRA M	Exclusive Or memory with A	1	0	1	0	1	1	1	0	7
ORA M	Or memory with A	1	0	1	1	0	1	1	0	7
CMP M	Compare memory with A	1	0	1	1	1	1	1	0	7
ADI	Add immediate to A	1	1	0	0	0	1	1	0	7
ACI	Add immediate to A with carry	1	1	0	0	1	1	1	0	7
SUI	Subtract immediate from A	1	1	0	1	0	1	1	0	7
SBI	Subtract immediate from A with borrow	1	1	0	1	1	1	1	0	7
ANI	And immediate with A	1	1	1	0	0	1	1	0	7
XRI	Exclusive Or immediate with A	1	1	1	0	1	1	1	0	7
ORI	Or immediate with A	1	1	1	1	0	1	1	0	7
CPI	Compare immediate with A	1	1	1	1	1	1	1	0	7
RLC	Rotate A left	0	0	0	0	0	1	1	1	4
RRC	Rotate A right	0	0	0	0	1	1	1	1	4
RAL	Rotate A left through carry	0	0	0	1	0	1	1	1	4
RAR	Rotate A right through carry	0	0	1	1	1	1	1	1	4
JMP	Jump unconditional	1	1	0	0	0	0	1	1	10
JC	Jump on carry	1	1	0	1	1	0	1	0	7/10
JNC	Jump on no carry	1	1	0	1	0	0	1	0	7/10
JZ	Jump on zero	1	1	0	0	1	0	1	0	7/10
JNZ	Jump on no zero	1	1	0	0	0	0	1	0	7/10
JP	Jump on positive	1	1	1	1	0	0	1	0	7/10
JM	Jump on minus	1	1	1	1	1	0	1	0	7/10
JPE	Jump on parity even	1	1	1	0	1	0	1	0	7/10
JPO	Jump on parity odd	1	1	1	0	0	0	1	0	7/10
CALL	Call unconditional	1	1	0	0	1	1	0	1	18
CC	Call on carry	1	1	0	1	1	1	0	0	9/18
CNC	Call on no carry	1	1	0	1	0	1	0	0	9/18
CZ	Call on zero	1	1	0	0	1	1	0	0	9/18
CNZ	Call on no zero	1	1	0	0	0	1	0	0	9/18
CP	Call on positive	1	1	1	1	0	1	0	0	9/18
CM	Call on minus	1	1	1	1	1	1	0	0	9/18
CPE	Call on parity even	1	1	1	0	1	1	0	0	9/18
CPO	Call on parity odd	1	1	1	0	0	1	0	0	9/18
RET	Return	1	1	0	0	1	0	0	1	10
RC	Return on carry	1	1	0	1	1	0	0	0	6/12
RNC	Return on no carry	1	1	0	1	0	0	0	0	6/12
RZ	Return on zero	1	1	0	0	1	0	0	0	6/12
RNZ	Return on no zero	1	1	0	0	0	0	0	0	6/12
RP	Return on positive	1	1	1	1	0	0	0	0	6/12
RM	Return on minus	1	1	1	1	1	0	0	0	6/12
RPE	Return on parity even	1	1	1	0	1	0	0	0	6/12
RPO	Return on parity odd	1	1	1	0	0	0	0	0	6/12
RST	Restart	1	1	A	A	A	1	1	1	12
IN	Input	1	1	0	1	1	0	1	1	10
LXI B	Load immediate register Pair B & C	0	0	0	0	0	0	0	1	10
LXI D	Load immediate register Pair D & E	0	0	0	1	0	0	0	1	10
LXI H	Load immediate register Pair H & L	0	0	1	0	0	0	0	1	10
LXI SP	Load immediate stack pointer	0	0	1	1	0	0	1	1	10
PUSH B	Push register Pair B & C on stack	1	1	0	0	0	1	0	1	12
PUSH D	Push register Pair D & E on stack	1	1	0	1	0	1	0	1	12
PUSH H	Push register Pair H & L on stack	1	1	1	0	0	1	0	1	12
PUSH PSW	Push A and Flags on stack	1	1	1	1	0	1	0	1	12
POP B	Pop register Pair B & C off stack	1	1	0	0	0	0	0	1	10
POP D	Pop register Pair D & E off stack	1	1	0	1	0	0	0	1	10
POP H	Pop register Pair H & L off stack	1	1	1	0	0	0	0	1	10
POP PSW	Pop A and Flags off stack	1	1	1	1	0	0	0	1	10
STA	Store A direct	0	0	1	1	0	0	1	0	13
LDA	Load A direct	0	0	1	1	1	0	1	0	13
XCHG	Exchange D & E, H & L Registers	1	1	1	0	1	0	1	1	4
XTHL	Exchange top of stack, H & L	1	1	1	0	0	0	1	1	16
SPHL	H & L to stack pointer	1	1	1	1	1	0	0	1	6
PCHL	H & L to program counter	1	1	1	0	1	0	0	1	6
DAD B	Add B & C to H & L	0	0	0	0	1	0	0	1	10
DAD D	Add D & E to H & L	0	0	0	1	1	0	0	1	10
DAD H	Add H & L to H & L	0	0	1	0	1	0	0	1	10
DAD SP	Add stack pointer to H & L	0	0	1	1	1	0	0	1	10
STAX B	Store A indirect	0	0	0	0	0	0	1	0	7
STAX D	Store A indirect	0	0	0	1	0	0	1	0	7
LDAX B	Load A indirect	0	0	0	0	1	0	1	0	7
LDAX D	Load A indirect	0	0	0	1	1	0	1	0	7
INX B	Increment B & C registers	0	0	0	0	0	0	1	1	6
INX D	Increment D & E registers	0	0	0	1	0	0	1	1	6
INX H	Increment H & L registers	0	0	1	0	0	0	1	1	6
INX SP	Increment stack pointer	0	0	1	1	0	0	1	1	6
DCX B	Decrement B & C	0	0	0	0	1	0	1	1	6
DCX D	Decrement D & E	0	0	0	1	1	0	1	1	6
DCX H	Decrement H & L	0	0	1	0	1	0	1	1	6
DCX SP	Decrement stack pointer	0	0	1	1	1	0	1	1	6
CMA	Complement A	0	0	1	0	1	1	1	1	4
STC	Set carry	0	0	1	1	0	1	1	1	4
CMC	Complement carry	0	0	1	1	1	1	1	1	4
DAA	Decimal adjust A	0	0	1	0	0	1	1	1	4
SHLD	Store H & L direct	0	0	1	0	0	0	1	0	16
LHLD	Load H & L direct	0	0	1	0	1	0	1	0	16
EI	Enable interrupts	1	1	1	1	1	0	1	1	4
DI	Disable interrupts	1	1	1	1	0	0	1	1	4
NOP	No operation	0	0	0	0	0	0	0	0	4
RIM	Read Interrupt Mask	0	0	1	0	0	0	0	0	4
SIM	Set Interrupt Mask	0	0	1	1	0	0	0	0	4

NOTES : 1. DDD or SSS - 000 B - 001 C - 010 D - 011 E - 100 H - 101 L - 110 Memory - 111 A.
2. Two possible cycle times, (6/12) indicate instruction cycles dependent on condition flags.

8080 et 8085

Les programmes rédigés pour le 8080 «tourneront» sur le 8085 avec lequel il est compatible, ce dernier comprenant cependant des instructions (SIM et RIM) et des possibilités supplémentaires, entrées-sorties en série, par exemple, ou d'interruptions vectorisées 7,5, ou autres.

Le moniteur se réserve l'usage d'une partie de la mémoire avec, en particulier, les affectations suivantes à quelques adresses :

Version 2-1	Version 1-2	Destination
20C2	20C8	3 cellules pour RST 5 3 cellules pour RST 6 3 cellules pour RST 6,5 3 cellules pour RST 7 3 cellules pour RST 7,5 } Disponibles pour l'utilisateur
20C5	20CB	
20C8	20CE	
20CB	20D1	
20CE	20D4	
20D1 à	20D7 à	
20E8	20E8	} Pile moniteur (mémoire tampon)
20E9 à	20FF	
		Stockage de données permettant le bon fonctionnement des programmes moniteur (single-step en particulier).

Pour connaître la version du moniteur, il suffit de lire le contenu de la case mémoire 0029 : on lira C8 pour la version 1, C2 pour la version 2 (l'instruction RST 5 provoque un appel de sous-programme commençant en 0028 ; on trouve dans le moniteur à l'adresse 0028 un JUMP en RAM).

INSTRUCTIONS DU 8085

Quartet bas

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP	LXI B	STAX B	INX B	INR B	DCR B	MVI B	RLC		DAD B	LDAX B	DCX B	INR C	DCR C	MVI C	RRC
1		LXI D	STAX D	INX D	INR D	DCR D	MVI D	RAL		DAD D	LDAX D	DCX D	INR E	DCR E	MVI E	RAR
2	RIM	LXI H	SHLD	INX H	INR H	DCR H	MVI H	DAA		DAD H	LHLD	DCX H	INR L	DCR L	MVI L	CMA
3	SIM	LXI SP	STA	INX SP	INR M	DCR M	MVI M	STC		DAD SP	LDA	DCX SP	INR A	DCR A	MVI A	CMC
4	MOV BB	MOV BC	MOV BD	MOV BE	MOV BH	MOV BL	MOV BM	MOV BA	MOV CB	MOV CC	MOV CD	MOV CE	MOV CH	MOV CL	MOV CM	MOV CA
5	MOV DB	MOV DC	MOV DD	MOV DE	MOV DH	MOV DL	MOV DM	MOV DA	MOV EB	MOV EC	MOV ED	MOV EE	MOV EH	MOV EL	MOV EM	MOV EA
6	MOV HB	MOV HC	MOV HD	MOV HE	MOV HH	MOV HL	MOV HM	MOV HA	MOV LB	MOV LC	MOV LD	MOV LE	MOV LH	MOV LL	MOV LM	MOV LA
7	MOV MB	MOV MC	MOV MD	MOV ME	MOV MH	MOV ML	HLT	MOV MA	MOV AB	MOV AC	MOV AD	MOV AE	MOV AH	MOV AL	MOV AM	MOV AA
8	ADD B	ADD C	ADD D	ADD E	ADD H	ADD L	ADD M	ADD A	ADC B	ADC C	ADC D	ADC E	ADC H	ADC L	ADC M	ADC A
9	SUB B	SUB C	SUB D	SUB E	SUB H	SUB L	SUB M	SBB A	SBB B	SBB C	SBB D	SBB E	SBB H	SBB L	SBB M	SBB A
A	ANA B	ANA C	ANA D	ANA E	ANA H	ANA L	ANA M	ANA A	XRA B	XRA C	XRA D	XRA E	XRA H	XRA L	XRA M	XRA A
B	ORA B	ORA C	ORA D	ORA E	ORA H	ORA L	ORA M	ORA A	CMP B	CMP C	CMP D	CMP E	CMP H	CMP L	CMP M	CMP A
C	RNZ	POP B	JNZ	JMP	CNZ	PUSH B	ADI	RST0	RZ	RET	JZ		CZ	Call	ADI	RST1
D	RNC	POP D	JNC	OUT	CNC	PUSH D	SUI	RST2	RC		JC	IN	CC		SBI	RST3
E	RPO	POP H	JPO	XTHL	CPO	PUSH H	ANI	RST4	RPE	PCHL	JPE	XCHG	CPE		XRI	RST5
F	RP	POP PSW	JP	DI	CP	PUSH PSW	ORI	RST6	RM	SPHL	JM	EI	CM		CPI	RST7

UTILISATION DU KIT SDK 85

I - Écriture et lecture en mémoire : programme de chargement de l'accumulateur

Le SDK-85 est connecté à son alimentation et mis en service. On appuie sur



et on lit sur les afficheurs :



Le micro-ordinateur est désormais prêt.

Les cellules de sa mémoire RAM n'ayant pas été chargées, elles contiennent à ce moment des valeurs purement aléatoires, dues au hasard de la commutation des circuits électroniques. On va enregistrer des données («écrire la mémoire»), puis relire ce qui a été écrit pour bien vérifier qu'on peut rappeler les informations.

On va en profiter pour proposer un mini-programme, qui assure le chargement de la valeur 4F (hexadécimal) dans la mémoire. Il ne comporte qu'une instruction d'exécution, suivie par une seconde instruction «rendant la main», c'est-à-dire retournant la maîtrise des opérations au programme moniteur qui, sagement, attendra les ordres suivants.

En langage «clair», les instructions sont :

1. Charger 4FH dans l'accumulateur,
2. Retour au moniteur.

En langage d'assemblage, on le codera par :

1. MVI A, 4FH
2. RST 1

Ce qui signifie : MVI est mis pour «Move Immediate», soit charger la valeur qui suit (c'est un adressage dit immédiat), dans l'accumulateur, noté A, la valeur 4F hexadécimale. L'ordre RST 1 est un «Restart», qui renvoie au moniteur. Ce dernier, lorsqu'on exécutera ce programme, montrera qu'il a bien repris en main les opérations en affichant à nouveau «— 8085».

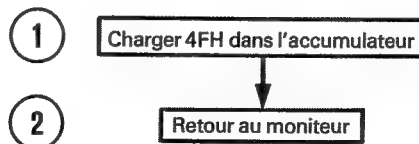
Si l'on se reporte aux instructions du 8085, données dans les tableaux, on voit que :

MVI A se code 3E ;
4FH se code 4F (c'est une donnée !) ;
RST 1 se code CF ;

Par conséquent, et puisque l'adresse de départ du programme est obligatoirement 2000 (hexadécimal) en raison de l'organisation de SDK-85, le plan d'occupation de la RAM est le suivant :

Adresse de la cellule	Contenu
2000	3E
2001	4F
2002	CF

Le programme complet est rédigé sur la feuille de codage (ci-contre) où toutes ces informations ont été regroupées, y compris des «Commentaires» qui permettent au programmeur de noter en langage clair ce qu'il a voulu faire ; ainsi pourra-t-on plus facilement comprendre le programme. On notera que sur cette feuille, chaque ligne correspond à une instruction complète. La colonne «Label» n'a pas été utilisée.



Adresse		Code			Mnémonique			Commentaires
page	ligne	1er octet	2ème octet	3ème octet	label	opération	Opérande ou adresse	
20	00	3E	4F			MVI	A,4FH	Charger 4F dans l'accumulateur
	02	CF				RST	1	Retour au moniteur

Introduction du programme en mémoire

Après avoir fait la remise à zéro, on lit — 8085 sur l'affichage. On va représenter par des carrés les actions sur les touches, et par un rectangle l'affichage et ce qu'il doit contenir *après* action sur la touche. Un point d'interrogation ou un X indique un contenu imprévisible, aléatoire. Ainsi, on fait :



Le moniteur attend les ordres... On va lui indiquer qu'on veut substituer aux contenus (aléatoires ou non) de la mémoire des valeurs précises. On presse le bouton SUBST MEM. L'affichage s'éteint et ne laisse plus subsister qu'un point qui indique que des valeurs vont être introduites *à sa gauche* ; on dispose alors des 4 afficheurs les plus à gauche, de quoi former ensuite une adresse. Cette étape est résumée par ;



On va maintenant former cette adresse en pressant successivement les touches 2, puis 0, à nouveau 0, et enfin le dernier zéro, ce qui donne (le point est toujours présent) :

2	0002.
0	0020.
0	0200.
0	2000.

L'adresse de départ est formée. Vérifiez que vous ne vous êtes pas trompé, que c'est bien 2000 : en cas d'erreur, continuez à frapper les touches pour former 2000. Dès que c'est fait, *vous confirmez* en appuyant sur NEXT. Les deux digits de droite vont s'allumer et présenter une valeur aléatoire ; parce qu'elle est imprévisible, on va mettre des points d'interrogation ou des X à sa place. Le point s'est déplacé à l'extrémité de droite, indiquant que le moniteur attend des *données* qu'il affichera sur les deux digits de plus faible poids (à droite, donc). Le premier octet de la première instruction étant 3E, on va l'introduire. La succession des opérations est :

NEXT	2000 XX.
3	2000 03.
E	2000 3E.

Si vous avez commis une erreur en introduisant autre chose que 3E, continuez à frapper les touches, mais les bonnes, cette fois ! L'affichage voulu étant obtenu, *on confirme à nouveau* en enfonçant NEXT. A ce moment, l'affichage fait apparaître l'adresse suivante, soit 2001 : le compteur ordinal s'est incrémenté automatiquement, on va frapper le second octet de la première instruction, soit 4F, qu'on confirme et expédie en mémoire à l'adresse 2001 grâce à NEXT, puis on forme l'octet de la seconde et dernière instruction, soit CF, qu'on confirme et envoie en mémoire à l'adresse 2002 :

NEXT	2001 XX.
4	2001 04.
F	2001 4F.
NEXT	2002 XX.
C	2002 0C.
F	2002 CF.
NEXT	2000 XX.

Le programme est désormais en mémoire. Appuyons à nouveau sur RESET, le moniteur reprend la main :

RESET

- 80 85

Lecture de la mémoire

Les cellules de la mémoire vont conserver ce programme, tant qu'on n'interrompra pas l'alimentation tout du moins. On peut le vérifier en appelant les adresses successives. Faites-le, à partir de :

- 80 85

Il faut appeler l'adresse 2000, ce que l'on fait comme précédemment :

SUBST
MEM

.

2

0002.

0

0020.

0

0200.

0

2000.

Quel est le contenu de la cellule 2000 ? On va le savoir en frappant NEXT :

NEXT

2000 3E.

En frappant encore NEXT, on lit successivement :

NEXT

2001 4F.

NEXT

2002 CF.

Par conséquent, le contenu des cellules est bel et bien conforme à ce qu'on leur a confié. En frappant RESET, puis SUBST MEM, on peut former toute autre adresse que 2000, et par exemple 2002 :

RESET

- 80 85

SUBST
MEM

.

2

0002.

0

0020.

0

0200.

2

2002.

L'action sur NEXT fera apparaître le contenu de la cellule 2002 :

NEXT

2002 EF.

Ceci permet de viser une cellule quelconque et, si on le veut, d'en modifier le contenu.

Exécution d'un programme

Pour exécuter ce programme, il faut rendre la main au moniteur :

RESET

- 80 85

puis lui indiquer l'adresse de départ pour exécution. Attention, maintenant on va lui dire d'*aller à* (GO) l'adresse 2000, soit :

GO

XXXX. XX

2

0002.

0

0020.

0

0200.

0

2000.

L'exécution est lancée avec la commande EXEC. Elle est tellement rapide (quelques microsecondes) qu'aussitôt, le moniteur reprend la main et affiche son message d'attente d'ordres :

EXEC

- 80 85

Si tout s'est passé comme espère l'accumulateur doit contenir 4F. Vérifions-le en examinant le contenu de ce registre grâce à la touche «Examine Register» :

EXAM
REG

.....

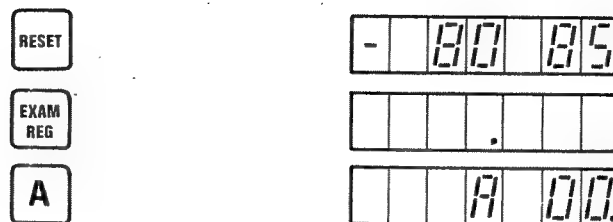
L'affichage s'éteint, à l'exception du point. Il reste à indiquer quel registre est en cause ; l'accumulateur étant désigné par A, on va frapper cette touche. Un A va s'afficher, avec le contenu de l'accumulateur sur les deux digits de plus faible poids, suivis d'un point :

A

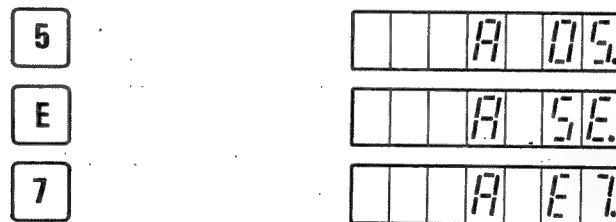
..A 4F.

L'accumulateur contient bien 4F.

Si l'on appuie sur RESET, le moniteur remet l'accumulateur à zéro et revient en position d'attente. On peut le vérifier en faisant :



On peut imposer un nouveau contenu à ce registre en frappant d'autres digits. Chaque nouveau digit se substitue à celui de plus faible poids qu'il chasse vers la gauche, le second digit étant perdu. Par exemple :

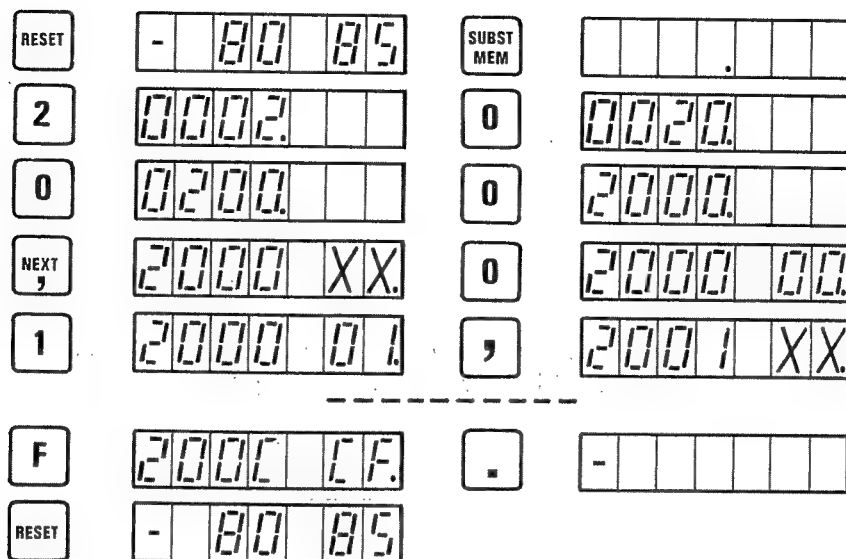


II - Mise au point d'un programme

A titre d'exemple nous reprenons le premier programme de cet ouvrage : la multiplication décimale de deux nombres, de un chiffre, différents de zéro.

Entrée du programme

Nous figurons les touches pressées, X, et l'affichage. Nous prenons $a = 05$ et $b = 04$.



On peut relire le programme pour voir s'il n'y a pas d'erreur.

Exécution du programme

GO XXXX8XX on tape 2000 comme précédemment :



On peut examiner D puisque l'affichage nous indique que l'exécution du programme est terminée.

EXAM REG

			.		
--	--	--	---	--	--

 D

			13		14
--	--	--	----	--	----

Pas à pas ou Single Step

RESET

-		80		85	
---	--	----	--	----	--

 STEP

X	X	X	X		X
---	---	---	---	--	---

On tape 2000

2	0	0	0		
---	---	---	---	--	--

 ,

2	0	0	3		16
---	---	---	---	--	----

La première instruction a été exécutée, on peut voir ce qu'elle a fait, on prévient le microprocesseur que «S.S.» est terminé et on examine les registres par :

•

-					
---	--	--	--	--	--

EXAM REG

			.		
--	--	--	---	--	--

 A

			A		00
--	--	--	---	--	----

,

			b		84
--	--	--	---	--	----

 ,

			C		05
--	--	--	---	--	----

•

-					
---	--	--	--	--	--

 STEP

			2003		16
--	--	--	------	--	----

etc... en examinant A,B, C, D à chaque instruction on obtient le tableau que nous avons vu.

En pas-à-pas ne jamais «repasser» par la case départ :

RESET

 qui fait perdre le contexte. Si cela vous arrive vous pouvez reprendre le programme en pas-à-pas à partir de *n'importe* qu'elle instruction à condition de charger tous les registres avec les valeurs lues avant la «fausse manœuvre». N'oubliez pas le registre F qui contient les flags.

Remarques

- on peut interrompre le pas à pas en pressant

GO

 après

EXEC

 , puis

EXEC

- nous avons fait figurer

,

 à la place de

NEXT

 et

.

 à la place de

EXEC

 pour préciser le rôle des touches

,

 sépare les «mots»

.

 termine la phrase.

Table des matières

Préambule	5
Introduction	6
Pour mieux suivre les listages	6
Le matériel	7
1 — Multiplication décimale de deux nombres de un chiffre (différents de zéro) .	15
2 — Conversion DCB - hexadécimal pour un nombre de deux chiffres,	19
3 — Multiplication de deux nombres de deux chiffres	24
4 — Affichage	26
5 — Le circuit d'interface 8279	27
6 — Entrée de données au clavier	30
7 — Addition en hexadécimal avec affichage du 1er opérande ou du résultat ..	33
8 — Addition avec affichage simultané du cumulande et de la somme	35
9 — Addition en décimal de deux fois deux digits avec résultat affiché sur trois digits	36
10 — L'inventaire	39
11 — Somme des n premiers nombres	42
12 — Le plus grands de deux nombres	45
13 — Entrée d'un nombre de deux chiffres avec affichage	48
14 — Multiplication de deux nombres positifs de deux chiffres, entrés au clavier. Le résultat est affiché.	50
15 — Multiplication en hexadécimal de deux nombres entiers négatifs ou positifs	53
16 — Addition sur 16 bits	57
17 — Conversion hexadécimal - BCD pour nombres entiers	59
18 — Conversion BCD - hexadécimal pour nombres entiers	61
19 — Conversion BCD - hexadécimal et hexadécimal-BCD pour nombres non entiers	65
20 — Division en hexadécimal d'un nombre de 4 chiffres par un nombre de 2 chiffres, avec virgule au résultat	72
21 — Multiplication en hexadécimal avec virgule	75
22 — Temporisations	77
23 — Codage des touches d'un clavier	79
24 — Insertion d'un complément au programme	82
25 — Chargement de tables	84
26 — Adressage de tables par calcul d'adresse	88
27 — Adressage de tables par recherche de la donnée, conversion : hexadécimal - 7 segments	90
28 — Utilisation de la touche VECT-INTR	93
29 — Exercices avec le 8279	98
30 — Affichage séquentiel avec effacement puis entrée en mémoire	104

31 — Chenillard (journal lumineux)	106
32 — Entrées-sorties du kit SDK 85 : carrefour	108
33 — Génération d'une note	114
34 — Réalisation d'un «piano»	117
35 — Boîte à musique	119
36 — Carillon de porte	121
37 — Serrure électronique	124
38 — Tirage du loto	127
39 — Création d'un nombre	130
40 — Horloge 24 heures avec sonnerie	131
41 — Bataille navale	135
42 — Jeu de NIM	139
43 — Jeu de Marienbad	143
44 — Master Mind	147
45 — Algorithme de tri	152
46 — Calcul et introduction de la parité	155
47 — Conversion : digital - analogique et générateur de fonctions	157
48 — Conversion : analogique - digital et voltmètre digital	159
49 — Conversion : analogique - digital par approximations successives	163
50 — Tir au pigeon	165
51 — Programmeur de mémoire EPROM	167
Le kit SDK 85	172
Le matériel : le microprocesseur 8085 et le SDK 85	172
8080 et 8085	180
Utilisation du KIT SDK 85	181

S.E.C.F.



EDITIONS RADIO

Service lecteurs

(à retourner à S.E.C.F.-Éditions Radio, 9, rue Jacob, 75006 Paris)

Pour nous permettre de vous proposer des ouvrages toujours meilleurs, nous souhaiterions recevoir vos critiques, appréciations et suggestions sur le présent livre :

Quels sont les ouvrages (thème, sujet, niveau) que vous souhaiteriez voir publier par notre société ?

Nous vous remercions de votre confiance et de votre coopération.

S.E.C.F.-Éditions Radio

Je désire recevoir gratuitement et sans engagement (mettre une croix dans la case) :

- ☐ Votre catalogue général (Electronique professionnelle et grand public, Informatique, Hi-Fi, Vidéo)
☐ Votre catalogue spécial informatique.

Nom : _____ Prénom : _____

Adresse : _____

Secteur d'activité et fonction : _____

CENTRES D'INTÉRÊTS

- | | |
|---|---|
| <input type="checkbox"/> Electronique professionnelle | <input type="checkbox"/> Micro-informatique professionnelle |
| <input type="checkbox"/> Electronique de loisirs | <input type="checkbox"/> Micro-informatique de loisirs |
| <input type="checkbox"/> Vidéo | <input type="checkbox"/> Autres : |
| <input type="checkbox"/> Hifi, CB... | |

Voir au dos "Correspondance Auteurs"

INITIATION AU LANGAGE ASSEMBLEUR

Le langage assembleur est le plus efficace qui soit. Vous l'utiliserez de préférence au langage évolué, chaque fois qu'il vous faudra rédiger des programmes occupant le moins de mémoire possible et devant être rapidement exécutés. Ou encore, lorsque votre application présente des exigences très particulières : applications scientifiques ou industrielles, par exemple.

Ce livre vous enseigne comment programmer en assembleur au travers d'une cinquantaine d'exercices-programmes de difficulté graduée. Il s'appuie sur la famille des microprocesseurs 8080 ; 8085 ; Z80 ; MCS 800 ; etc. Il vous permettra d'aborder la programmation en assembleur aussi bien sur des micro-ordinateurs industriels, que sur des machines standard de bureau.

S. E. C. F.



ÉDITIONS RADIO



9 782709 109352

ISBN 2 7091 0935 2
Code 1

Prix : 130 F